Power of... DOS 5.0

THE COMPLETE GUIDE

TOM GOODELL

MIS: PRESS

Power of...

DOS 5.0

TOM GOODELL

MIS: PRESS

A Subsidiary of
Henry Holt and Co., Inc.

Copyright © 1991 by Management Information Source, Inc.
a subsidiary of Henry Holt and Company, Inc.
115 West 18th Street
New York, New York 10011

All rights reserved. Reproduction or use of editorial or pictorial content in any manner is prohibited without express permission. No patent liability is assumed with respect to the use of the information contained herein. While every precaution has been taken in the preparation of this book, the publisher assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained herein.

First Edition—1991

ISBN 1-55828-148-7

Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

MIS:Press books are available at special discounts for bulk purchases for sales promotions, premiums, fund-raising, or educational use. Special editions or book excerpts can also be created to specification.

For details contact:
Special Sales Director
MIS:Press
a subsidiary of Henry Holt and Company, Inc.
115 West 18th Street
New York, New York 10011

TRADEMARKS

Clipper is a trademark of Nantucket Corporation
Compaq is a trademark of Compaq Corporation
dBASE is a trademark of Ashton-Tate
DESQview and QEMM-386 are trademarks of Quarterdeck Office Systems
Epson is a trademark of Seiko Epson Corporation
FoxBASE is a trademark of Fox Software
IBM PC, XT, AT, PC-DOS, MDA, CGA, MGA, EGA, MCGA, VGA, 8514/A, and XGA are trademarks of IBM Corporation
Intel 8088, 8086, 80286, 80386, 80387, 80486, and 80586 are trademarks of Intel Corporation
Lotus 1-2-3 is a trademark of Lotus Development Corporation
Mace Utilities is a trademark of Fifth Generation Systems
Microsoft C is a trademark of Microsoft Corporation
Microsoft Excel is a trademark of Microsoft Corporation
Microsoft Word and Microsoft Windows are trademarks of Microsoft Corporation
MS-DOS is a trademark of Microsoft Corporation
The Norton Utilites is a trademark of Symantec Corporation
PC Tools is a trademark of Central Point Software, Inc.
Q Basic and Quick Basic are trademarks of Microsoft Corporation
Quattro Pro is a trademark of Borland International, Inc.
R:BASE is a trademark of Microrim, Inc.
Reflex is a trademark of Borland International, Inc.
SideKick is a trademark of Borland International, Inc.
Symphony is a trademark of Lotus Development Corporation
WordPerfect is a trademark of WordPerfect Corporation

Dedication

This book is for Sarah, who keeps reminding me of life when I get lost in work.

EDINBURGH CITY LIBRARIES
AC C0004078349
LY CRD
CL QA 76 .76 O63
PR £22.50
V 2014 9L 37
X

Acknowledgements

I would like to thank the following people for their help and support in the writing of this book:

Lily Lasker, who worked long evenings and weekends preparing the manuscript;

Theresa Wolner of Information Lost & Found Indexing Services, who provided invaluable advice for index preparation;

The staff of MIS:Press, who as always provided expert advice and direction;

My wife Barbara and daughter Sarah, who arranged their days and nights around my writing schedule;

The clients of Integrated Knowledge Systems, Inc. who continue to present me with challenging problems.

Contents

INTRODUCTION 1

Additional Information 4

CHAPTER 1: UNDERSTANDING PERSONAL COMPUTERS AND DOS 5

How Personal Computers are Organized 7

The Central Processing Unit (CPU) 8

The Monitor 10

The Keyboard 13

Disk Drives 15

Electronic Memory 18

DOS 19

CHAPTER 2: THE DOS 5.0 SHELL 23

Overview of the Shell 24

Navigating in the Shell 26

Using Scroll Bars 27

Changing the View 28

Selecting Disks 29

Exiting the Shell 30

Using Shell Menus and Dialog Boxes 31

Activating the Menu Bar 36

Getting Online Help 36

Managing Directories in the Shell 39

Selecting a Directory 42

Creating Directories 42

Deleting Directories 43

Renaming Directories 44

Expanding and Collapsing Subdirectories 45

Managing Files in the Shell 46

Displaying Technical Information About a File 50

Working with Files in the Shell 51
- Selecting Files 53
- Associating Data Files with Program Files 56
- Opening Files 59
- Printing Files from the Shell 59
  - Print queue 60
  - Printing with copy 62
- The Search Option 63
- The View Option 64
- Moving Files from One Directory to Another 68
- Copying Files 72
- Deleting Files 74
- Renaming Files 75
- Changing File Attributes 77
- Undeleting Files 79

Running Programs in the Shell 84
- … from the File List Area 84
- … from the RUN Option 85
- … from the Program List Area 85

Managing the Program List Area 86
- Adding a Program to a File List Menu 88
- Modifying a Program in a Program List Menu 92
- Defining an Application Shortcut Key 92
- Pause After Exit Option 93
- Adding a Password to a Program 93
- Recovery from Lost Passwords 94
- Providing Online Help for Programs 97
- Specifying Memory Requirements for Programs 99
- Setting the Video Mode for a Program 101
- Disabling the Shell's Shortcut Keys 101
- Preventing the Task Swapper from Switching a Program 101
- Adding a Group to a Program List Menu 101
- Writing Custom Shell Programs 104
- Copying Programs from One Group to Another 105
- Changing the Order of Items within a Group 106
- Deleting Programs and Groups 107

The Task Swapper – Running More than One Program at Once 108
Turning the Task Swapper On and Off 109
Adding and Removing Programs in the Active Task List 111
Navigating in the Task Swapper 112
Handling Programs that Lock Up 113
Managing the Shell Environment 114
Redrawing and Updating the Screen 114
Changing Screen Colors 114
Text and Graphics Modes 115
Controlling the Display of Confirmation Dialog Boxes 116
Specifying Startup Options 118
Advanced Shell Topics 119
Understanding the Move File Operation 119
Structure of the DOSSHELL.INI File 121
Developing Advanced Help Screens 139
Advanced Color Adjustments 144
CHAPTER 3: THE DOS COMMAND LINE 149
Understanding the Path 151
Syntax of Commands 153
Using the DOS Command Line 155
Shortcuts for Re-entering and Editing Commands 156
Obtaining Online Help for DOS Commands 162
Pausing and Stopping DOS Commands 165
Redirecting Command Line Input and Output 166
The DOS Filter Commands — MORE, FIND, and SORT 168
Frequently Used DOS Commands 176
CLS 176
DEL 176
DIR 176
DOSKEY 177
FOR 178
FORMAT 180
RENAME 181
TYPE 181
VER 181

Customizing the Command Prompt 182
Including a Dollar Sign in the Prompt 183
Changing the DOS Prompt to a Text Message 183
Including a Greater-than Sign (>) in the DOS Prompt 183
Including a Less-than Sign (<) in the DOS Prompt 184
Including the Current Drive Letter in the DOS Prompt 184
Including the Current Directory Path in the DOS Prompt 185
Including the Date in the DOS Prompt 186
Including the Time in the DOS Prompt 187
Including the DOS Version Number in the DOS Prompt 187
Including a Vertical Bar (Pipe) Symbol in the DOS Prompt 188
Including an Equal Sign in the DOS Prompt 188
Using a Backspace in the DOS Prompt 189
Displaying More than One Line in the DOS Prompt 189
Changing the Color of the Prompt 190
Using the Doskey Command Stack 191
Entering Multiple Commands on One Line 192
Recalling Previous Commands 192
Keys Used for Editing the Displayed Command 196
Creating Batch and Text Files from the Command Stack 198
Creating and Using Macros with Doskey 200
Creating Macros 200
Running Macros 201
Creating and Using Macros with Multiple Commands 202
Using Replaceable Parameters in Macros 203
Redirecting Macro Input and Output 205
Listing, Deleting, and Saving Macro Definitions 207
Using Dollar Sign Symbols in Macros 209
The Doskey Command Syntax 210
Advanced Topics 211
The DIRCMD Environment Variable 211
Doskey — How it Works 215
Doskey Memory Management 215

CHAPTER 4: UNDERSTANDING AND MANAGING MEMORY 217

Understanding Memory Resources 219

Bits and Bytes 220

How Memory is Organized 222

How DOS Uses the First 640K of Memory 224

Expanded Memory 228

Extended Memory 231

Managing Memory Resources 233

Maximizing the Use of Conventional Memory 233

Managing Expanded Memory 243

Managing Extended Memory 246

Memory Management Summary 249

Advanced Topics 250

Understanding Memory Addresses 250

Switch Settings for the EMM386.EXE Device Driver 251

Switch Settings for the HIMEM.SYS Device Driver 257

More on RAM 261

Real, Protected, and Enhanced Chip Modes 262

CHAPTER 5: UNDERSTANDING AND MANAGING FILES AND DIRECTORIES 265

Files 266

File Names 267

Directories 273

The TREE Command 275

The DIR Command 277

Changing the DIR Default Switches 285

Specifying Subdirectories 287

Printing and Saving Directory Listings 288

Organizing Directories 289

DOS Commands For File and Directory Management 290

Commands for Comparing, Copying, and Deleting Files 290

Renaming Files 317

Commands for Locating and Displaying Files 318

File Protection and Recovery Commands 320

Commands to Create, Delete, and Navigate in, Directories 346

Interrupting DOS Commands 352

Advanced Topics 353
How Files are Stored in Directories 353
Understanding the Root Directory 358

CHAPTER 6: UNDERSTANDING AND MANAGING DISKS 359
What Disks Are and What They Do 360
The Physical Structure of Disks 360
Types of Disks 361
The Logical Structure of Disks 363
Tracks, Cylinders, and Sectors 363
Interleave 366
Disk Performance 367
Formatting Disks 368
Disk Partitions 371
Managing Disks 372
FORMAT 374
UNFORMAT and MIRROR 379
FDISK — Partitioning a Hard Drive 386
SYS — Creating a System Disk 388
LABEL — Modifying the Volume Label 389
VOL — Displaying the Volume Label 390
CHKDSK — Checking Disk Integrity 391
DISKCOMP — Comparing Two Floppies 394
APPEND 395
ASSIGN — Reassigning Drive Letters 397
JOIN — Joining Two Drives 399
SUBST — Substituting Drive Letters 400
BUFFERS 401
FASTOPEN 405
SMARTDRIVE 407
RAMDRIVE — Speeds Up Your System 409
Disk Fragmentation 413
Adjusting the Interleave 414

CHAPTER 7: THE MS-DOS EDITOR 415
Starting and Leaving the MS-DOS Editor 416
Navigating in the Editor 417
The File Menu 418
Editing Files 421
Moving Around in a File 422
The Edit Menu 423
The Search Menu 424
The Options Menu 426
Adjusting the Colors and Screen Display 427
Setting the Location of Help Files 428
Using Online Help in the Editor 428

CHAPTER 8: BATCH FILES 431
Batch File Programs versus Shell Programs 432
Creating Simple Batch Files 433
Using Echo and Pause Commands in Batch Files 434
Parameters and Environment Variables in Batch Files 436
Creating Batch Files that Call Themselves 437
Halting the Execution of a Batch File 439
Use of the IF Command in Batch Files 439
Placing Comments in Batch Files 442
Developing Complex Batch Files 443
Branching of Program Control with the GOTO Command 444
Using FOR to Repeat Commands 446
Batch Files that Call Other Batch Files 450

CHAPTER 9: CONFIGURING AND CUSTOMIZING DOS 455
The CONFIG.SYS and AUTOEXEC.BAT Files 456
The DOS Environment 456
Creating Environment Variables with the SET Command 457
Changing the Environment Size with SHELL 458
Device Drivers 459
Using ANSI.SYS to Customize the Screen and Keyboard 460
PROMPT Parameters Used for Positioning the Cursor 461
PROMPT Parameters Used for Modifying the Screen Display 465
PROMPT Parameters Used for Modifying the Keyboard 470

Fine-tuning the Keyboard Response Rate with the MODE Command ... 473

Managing the Print Queue ... 473

Specifying the Printing Device ... 475

Specifying the Size of the Print Buffer ... 476

Specifying the Maximum Number of Files in the Queue ... 477

Managing PRINT Time ... 477

Managing Files in the Queue ... 480

Use of Wild Cards in Print Command File Names ... 483

Specifying the Maximum Number of Logical Drives ... 484

Controlling the Effects of Control-Break ... 485

CHAPTER 10: THE DOS COMMAND DICTIONARY ... 487

@ ... 488

APPEND ... 489

ASSIGN ... 490

ATTRIB ... 491

BACKUP ... 492

BREAK ... 494

BUFFERS ... 495

CALL ... 496

CD or CHDIR ... 498

CHKDSK ... 498

CLS ... 500

COMP ... 500

COPY ... 502

COUNTRY ... 505

DATE ... 506

DEL ... 507

DEVICE ... 508

DEVICEHIGH ... 508

DIR ... 509

DISKCOMP ... 510

DISKCOPY ... 511

DOS ... 512

DOSKEY ... 513

DOSSHELL ... 513

ECHO ... 513

EXPAND 514
FASTOPEN 515
FC 516
FILES 517
FIND 517
FOR 518
FORMAT 519
GOTO 520
GRAPHICS 521
HELP 522
IF 522
INSTALL 523
JOIN 523
KEYB 524
LABEL 526
LASTDRIVE 526
LOADHIGH or LH 527
MEM 528
MIRROR 530
MKDIR or MD 530
MORE 531
PATH 532
PAUSE 533
PRINT 533
PROMPT 534
RECOVER 535
REM 536
RENAME or REN 537
REPLACE 537
RESTORE 539
RMDIR or RD 542
SET 542
SETVER 543
SHELL 544
SHIFT 544
SORT 546

SUBST ... 547
SWITCHES ... 548
SYS ... 548
TIME ... 549
TREE ... 549
TYPE ... 550
UNDELETE ... 550
UNFORMAT ... 551
VER ... 552
VERIFY ... 552
VOL ... 553
XCOPY ... 553

APPENDIX: IBM EXTENDED CHARACTER SET ... 555

INDEX ... 563

Introduction

DOS 5.0 is a major step in the evolution of the DOS operating system for IBM-compatible Personal Computers. It adds numerous features and commands that permit far more control over the computer's use of memory and disks than any previous version. With this expanded power comes a price, however: to make full use of it, PC users will have to understand their computers better than ever before. This book is written with that need in mind.

The Power of...DOS 5.0 is intended for all users of DOS 5.0 who know at least the basics of how to use their computer. The material is presented "from the ground up," assuming little or no familiarity with how Personal Computers work on the inside. Only after establishing a clear understanding of the basics are the advanced topics covered. Novice users will want to start with the fundamental material at the beginning of each chapter, and some may never find it necessary to go any further. Advanced users may want to skip some of the introductory material and go right to the advanced topics.

While each chapter can stand alone, the sequence of the chapters within the book provides a logical structure for learning DOS 5.0. If your goal in reading this book is to move from a novice to an advanced stage of understanding, then you may want to read the chapters in order.

Briefly, the chapters cover the following material:

Chapter 1 provides an introduction to the fundamentals of personal computers, explaining the various parts of the computer and the role that DOS plays in making them all work together.

Chapter 2 discusses the DOS 5.0 Shell. The Shell provides easy, intuitive access to DOS and to the programs installed on your computer. The Shell is menu driven, so it alleviates the need to memorize DOS commands, and makes it much easier to manage the disks and files in your computer. The Shell also includes a "Task Switcher," which lets you run more than one program at once, and switch back and forth between them. This chapter covers using the Shell, and how to customize it to fit your own particular set of programs and work habits. Since the Shell provides access to many of the features of DOS 5.0 that are discussed in greater detail in subsequent chapters, this section also serves as an introduction to those features.

Chapter 3 provides a detailed explanation of the DOS command line, which is where you can enter DOS commands and run programs. Working at the command line gives you a finer level of control over your computer than the Shell provides.

Chapter 4 explains the functioning and management of the one of the most important resources available to a computer: electronic memory. How DOS 5.0 controls these resources, and how you can fine tune that control, is explained.

Chapter 5 discusses files, directories, and their management. Files are the repositories of all of the programs and data that your computer uses. Directories provide the structure for keeping them organized on your disks. DOS 5.0 provides numerous commands to facilitate the creation and management of these structures.

Chapter 6 explains the functioning and management of another major resource in the computer: disk storage. Disk storage is often treated as part of the subject of files and directories, but this tends to obscure the fact that there are distinct issues surrounding each of these topics, and distinct DOS commands for dealing with them. This chapter provides a thorough presentation of the inner workings of disks and disk drives, and of the DOS commands used to maximize their performance.

Chapter 7 teaches you how to use the MS-DOS Editor, the text editor that comes with DOS 5.0.

Chapter 8 explains BATCH files, which permit you to write programs using DOS commands. The creation and use of BATCH files is discussed in depth.

Chapter 9 discusses commands used to configure DOS to closely match your own working needs. As you use your computer and install new software and hardware on it, you will find that some programs and devices require special commands to be executed to configure the DOS environment properly. The kinds of settings you are likely to encounter and how they are installed is explained.

Chapter 10 provides a command dictionary for quickly looking up the syntax and usage of the commonly used DOS commands. While many of these commands are discussed in more detail elsewhere, this chapter is the quickest place to find a summary reference to them. The complete syntax of each command, a brief description of what it does, and a listing of parameters and switches, are given here. If you are familiar with the basics of the command, this chapter may contain all the information you need.

Each chapter is designed to lead you gradually from basic concepts to advanced material. Most chapters end with an Advanced Topics section, which discusses material that goes into more technical detail than is generally required for using DOS, but that is interesting to understand.

While the material in each chapter is inevitably related to the rest of the book, the chapters are designed to stand on their own as much as possible. The book does not have to be read in sequence; you may want to jump from one place to another, learning the various aspects of DOS as needs arise. For example, if all you want to do is to use the DOS shell to run programs and manipulate files, then you can skip directly to Chapter 2. However, if you are relatively new to Personal Computers, then you should read chapter one, which presents basic material essential to understanding the rest of the book.

DOS 5.0 is an exciting product; it breathes new life into a venerable operating system that promises to be the standard for Personal Computers for many more years. This book is presented in the spirit of helping you get the most use out of DOS 5.0, with the least effort.

Additional Information

When references are made to keys of the PC keyboard, such as Enter or Return, this book will use key-like icons, such as Enter←. If the keys need to represent a key combination, such as holding down the Alt key while pressing the F10 function key, it will be indicated as follows: alt-F10. Ctrl-alt-delete (which reboots your computer) would mean to hold down the Control key, and hold down the Alt key, and while holding down these two keys, press the Delete key.

Throughout the book there will be sections that require special attention. The following icons will be used to mark these sections.

LoOk	Indicates that you should take note of the information. This symbol may indicate a helpful hint or a special condition.
SHORTCUT	Indicates that you can perform an action more quickly by using shortcut keys or by following the suggestion in the text.
STOP	Indicates cautionary information or warnings. This symbol often provides a warning that you may lose data if you incorrectly perform an action.
WARNING	Indicates a serious warning. Data may be lost or the system may hang-up or crash. A misuse of these functions may seriously jeopardize your data, disk, or system operation.
GOOD IDEA	Indicates a particularly useful feature or usage. This is also used to indicate an exceptional suggestion for a procedure or process.

Chapter 1

Understanding Personal Computers and DOS

Computers are usually thought of as machines — collections of electronic and mechanical devices that interact with each other. This view only considers the hardware — the physical devices that make up the machine. A computer cannot accomplish anything without instructions, however, and there are even instructions that are built into the hardware. The instructions that tell a computer what to do are referred to as **software**, and the instructions that are built into the hardware are called **firmware**. It is most useful to think of a computer as the combination of hardware, software, and firmware. This chapter reviews the fundamentals of how a personal computer is organized, how it works, and what DOS 5.0 is and does. Readers who have a basic understanding of how Personal Computers work, and are familiar with an earlier version of DOS, may want to skim this chapter.

The words bits, bytes, kilobytes, and megabytes are used in many contexts when talking about computers, so it is important to have a clear understanding of what these mean. A computer is really just a huge collection of switches, each of which can either be off or on. The states of off and on correspond to the numbers 0 and 1. Consequently, all of the information in a computer is comprised of sequences of 0s and 1s — in other words, binary, or base two numbers. When operating on alphabetic characters, the computer represents the letter A by the number 65, which in base two is 01000001. So when an A appears on your screen, somewhere inside your computer is a set of switches in the following sequence: OFF ON OFF OFF OFF OFF OFF ON.

A single switch, or binary digit, is called a "bit." Since a single bit is such a small amount of information, bits are organized into groups of eight (notice that eight bits were used to represent the letter A above, even though the left most one was a zero and therefore not numerically relevant). A collection of eight bits is called a "byte." Bytes are sometimes also grouped together, into sets of two or four. These groups are called "words." A word of two bytes is called a 16-bit word, while a word of four bytes is called a 32-bit word. Information is represented inside the computer as one or more bytes. All alphabetic characters require a single byte. The largest number that can be represented by a single byte is 255, so to store numbers larger than 255, and for "real" numbers, two or more bytes are used.

Since computers work in base two numbers, their storage capacities are measured in powers of the number two. That is why we commonly see the numbers eight (2^3), sixteen (2^4), thirty-two (2^5), sixty-four (2^6), etc. Two to the tenth is 1024. This number is pretty close to 1,000, so the term "kilobyte," which literally means "1,000 bytes," is used to refer to 1024 bytes of information. Likewise, the term "megabyte" refers to 1024^2, which is 1,048,576, or about one million bytes. Rather than spell out kilobyte and megabyte, the terms K and M, or KB and Meg, are commonly used. Thus, 640K or 640 KB refer to 640 kilobytes; and 4M, or 4 Meg, refer to 4 megabytes.

How Personal Computers are Organized

The hardware that defines a Personal Computer is organized into five primary parts, or groups of parts: (1) the CPU, or Central Processing Unit, and related circuits; (2) a monitor, or screen; (3) a keyboard; (4) one or more disk drives; and (5) electronic memory. There can be other hardware devices attached to the computer as well, such as printers, mice, and modems. These are referred to as "peripheral devices."

Integral to the organization of the Personal Computer is the concept of "expansion slots." In order to make the Personal Computer flexible and adaptable, it was designed to allow various parts to be replaced and upgraded without having to replace the entire computer. The core elements of the machine — the CPU, electronic memory, and other vital circuits that are not likely candidates for upgrading — are placed on a single board, called the "motherboard," that is permanently installed in the computer case. The rest of the circuits go on cards that plug into the "expansion slots." Expansion slots are sockets on the motherboard, with space above them to accommodate circuit boards that plug into the sockets. When you hear reference to the "number of slots" a machine has, it means the number of expansion slots.

Expansion slots are used for a variety of purposes. If you want more electronic memory than can be stored on the computer's motherboard, you can install a card with additional memory in one of the expansion slots. The computer's disk drives and monitor both require special circuits to control them. These are placed on cards that plug into the expansion slots, making it easy to upgrade the disk drives and monitor without having to buy a new computer or replace the motherboard. You can even buy expansion cards that contain complete modems and hard drives.

Expansion slots come in different lengths, referred to as full length, three-quarters length, and half length. Often the internal design of a computer case does not leave room for a full length slot, but rather than waste that space, a shorter slot is provided. Not all expansion cards come in different lengths; some are only available in full-length versions.

The Central Processing Unit (CPU)

The CPU is the "brains" of the computer. Everything the computer does is directed by this chip (the word "chip" refers to integrated circuits, which are the devices that store electronic information and execute electronic instructions). There are two primary functions that the CPU carries out: (1) it directs information flow and coordinates the actions of other devices in the computer; and (2) it performs the actual computing steps involved in running software programs.

Since the CPU is the brains of the computer, it makes sense that it is the single most important hardware component — it defines the ultimate limits of what the computer can do. References to different types of IBM-compatible Personal Computers invariably make reference to the type of chip used for the CPU. The original IBM PC was introduced in 1981 and used a chip known as the 8088, manufactured by the Intel Corporation. The 8088 had a "big brother" named the 8086. Both chips were capable of 16-bit processing, which means that they could operate on information in chunks of 16 bits, or two bytes, at a time. These two chips were identical in all respects except one: the 8088 communicated with the rest of the computer in 8-bit, or 1-byte, chunks of information, while the 8086 communicated in 16-bit, or 2-byte, chunks. The 8088 was chosen for the first PC because it was less expensive to build a computer around an 8-bit communication path, or "bus," than a 16-bit bus. Since the first PC was built, there have been a number of generations in this family of chips, each providing more power and sophistication than the previous one. In subsequent redesigns, the 8086 evolved into the 80286 (16-bit), 80386 (32-bit), 80486 (32-bit), etc.

The first chip to follow the 8088/8086 was the 80286. The 80286 was used to build the IBM-AT Personal Computer. This chip, which increased the amount of memory the computer could use, also performed computing and internal communication operations at much faster speeds than the original PC. The next chip to come along was the 80386. The 80386 not only provided dramatically higher operating speeds, but also vastly increased the amount of memory the computer could address. It provided the ability to operate on 32-bit chunks of information, and went from a 16-bit bus to a 32-bit bus, effectively doubling the speed with which it could communicate with other parts of the computer. The 80486 provided yet another leap in processing speed and also had specialized circuitry built into it for very high-speed mathematical processing — an ability that previously required purchasing a special add-on chip (the 80x87). The 80586 is currently in development, and Intel has drawn up the first plans for the 80686. This family of chips will continue to evolve, providing more and more computing power at lower and lower prices. Since the names of all these chips begins with 80, they are usually referred to simply by the last three digits — hence the common references to a “286 computer” or a “386 computer.”

One of the significant features of the 80x86 line of chips is that they are all backward compatible. This means that all software that runs on 8088 machines will run on any PC that has one of these chips as its CPU. In other words, each chip in the line can run software designed for earlier chips. The chips are NOT forward compatible — software written to take advantage of the special features of the 80386 chip will not run on 8088 or 80286 machines. Most software is still designed to run on the entire line of chips, but it is becoming more common for software to be designed to only run on the 80386 or later CPU’s. In many cases, software developers will release two versions of their software — one which runs on any machine, and one which runs only on the higher-end machines.

Computer manufacturers and dealers typically refer to two aspects of the CPU when describing computers: the chip that the computer is built around, as discussed above, and the “clock speed” of that chip. Every computer has an internal clock — actually a specialized chip that creates electronic “ticks” at precise intervals. With each tick of the clock, the CPU moves forward one step in the sequence of instructions that it is executing. Consequently, the faster the clock, the faster the computer can carry out a given task. You cannot arbitrarily increase the clock speed of a computer, however, because every CPU has a maximum clock speed that it can handle.

The original IBM PC had a clock speed of 4.7 Megahertz (MHZ), or 4.7 million ticks per second. This seems like a lot, but the basic operations a CPU carries out are very simple. The commands we enter into the computer, and those that our software packages execute, are much more complex and are often broken down into hundreds or thousands of CPU operations. For much of today’s sophisticated software, a 4.7 MHZ machine is agonizingly slow.

As Intel developed new generations of chips in the 80x86 family, they also developed different versions of each generation. In most cases the difference between two versions of a chip is simply in the clock speed. The 80386 chip is available in 16 MHZ, 25 MHZ, and 33 MHZ versions. The 80486 is available in 25 MHZ and 33 MHZ versions, with faster versions in development.

As the 80386 chip caught on and software was developed to take advantage of it, Intel recognized an undeveloped market — users who wanted to run software designed for 32-bit processing, but who could not afford the cost of a 32-bit computer built around the 80386. Remember that the 80386 chip uses a 32-bit bus, or communication path. The larger the bus width, the more expensive the computer. So Intel developed another version of the 80386, named the 80386SX. The relationship between the 80386SX and the 80386DX (as the original version came to be called) is the same as that between the 8088 and the 8086 — the 80386SX uses a 16-bit bus, allowing manufacturers to build less expensive computers and still provide the 32-bit processing power of the 80386 chip.

The Monitor

There are really two hardware components to consider when discussing a monitor: the monitor itself, and the video card that drives it. There is usually nothing built into the computer itself that directly provides the output necessary to display an image on a monitor. That capability is provided by a video card. There is a good reason for this design: it allows you to upgrade your display without having to buy a new computer. As new and better monitor technologies have appeared, all that has been necessary to take advantage of them is to replace the video card and the monitor. When buying a video card and monitor, it is important to make sure they are well suited to each other — there is no point in buying a monitor that can display higher resolutions than the card can deliver, unless you intend to buy a better card later.

Monitor resolutions are described in terms of picture elements, or “pixels” for short. A pixel is the smallest area of the screen that the computer can display. Images, both text and graphics, are created as groups of pixels. The maximum resolution of a monitor is defined by the number of horizontal and vertical pixels that the monitor/video card combination is capable of displaying.

The original IBM PC had two options for the monitor: the MDA, or Monochrome Display Adapter, and the CGA, or Color Graphics Adapter. The MDA allowed only the display of text characters, which included a limited set of line-drawing characters. Its resolution was 720 pixels horizontally and 350 pixels vertically.

The CGA monitor allowed both text and graphics to be displayed, but the text was much lower resolution (320 x 200) than the Monochrome monitor provided and was therefore harder on the eyes. By today’s standards the graphics capabilities were crude. As is typical of graphics cards and monitors, it has several modes, each of which offers a specific resolution combined with a maximum number of colors that can be displayed at once. There is a trade-off between resolution and number of colors, because there is only a certain amount of memory that is available to store all of the information necessary to display an image. If more of that memory is used for pixels — which means higher resolution — then less is available for color information.

The MDA and CGA were the first video standards for IBM-compatible Personal Computers. Since that time there have been a total of seven additional standards developed, each one offering more resolution and/or more colors than the previous one. The history of PC video standards, and their capabilities, is shown in Table 1-1.

Table 1-1 — Evolution of Video Displays.

Name	Year	Resolution	Modes	Colors
MDA (Monochrome Display Adapter)	1981	720x350	Text	1
CGA (Color Graphics Adapter)	1981	640x200	Text	16
		320x200	Text	16
		160x200	Graphics	16
		320x200	Graphics	4
		640x200	Graphics	2
MGA (Hercules Mono. G.A.)	1982	720x350	Text	1
		720x348	Graphics	1
EGA (Enhanced Graphics Adapter)	1984	640x350	Text	16
		720x350	Text	4
		640x350	Graphics	16
		320x200	Graphics	16
		640x200	Graphics	16
		640x350	Graphics	16
MCGA (Multi Color Graphics Array)	1987	320x400	Text	4
		640x400	Text	2
		640x480	Graphics	2
		320x200	Graphics	256
VGA (Video Graphics Array)	1987	720x400	Text	16
		360x400	Text	16
		640x480	Graphics	16
		640x480	Graphics	2
		320x200	Graphics	256
8514/A	1987	1024x768	Graphics	16
		640x480	Graphics	256
		1024x768	Graphics	256
Super VGA	1989	800x600	Graphics	16
XGA (Extended Graphics Array)	1990	1056x400	Text	16
		640x480	Graphics	256
		1024x768	Graphics	256
		640x480	Graphics	65,536

When reviewing VGA and Super VGA monitors, you will often hear reference to "dot pitch." This is the size of the dots the monitor uses to display pixels. The smaller the dot pitch, the sharper the image.

As Table 1-1 shows, most video cards support a variety of modes. You can use DOS to shift the cards into their various modes, using the MODE command, which is discussed in Chapter 9.

The Keyboard

The keyboard is the most common device used to communicate with Personal Computers. The functioning of the keyboard is straightforward, but perhaps not quite as obvious as you might guess.

When you press a key on the keyboard, the corresponding character shows up on the screen, so it is natural to assume that, for example, the "A" key always means the letter "A." But in the spirit of keeping the computer flexible and adaptable, an additional step was inserted in the communication between the computer and the keyboard.

Every key on the keyboard has a "scan code." The scan codes are just numbers, starting with one and ending with the number of keys on the keyboard. When a key is pressed, the scan code for that key is sent to the computer, where DOS reads it and interprets the code to have a particular meaning — for example, the letter "A." Some keys are designed to be used in combination with other keys. These are referred to as Shift keys — they shift the keyboard to have a different set of meanings. For example, the "A" key can mean either upper or lower case A, depending on the state of the case-shifting key, ⇧Shift. So when the keyboard sends a scan code to the computer, it also sends the shift states of each of the Shift keys. DOS interprets each key differently depending on the state of the Shift keys.

The keyboard also sends a key-released code to the computer when a key is released. In this way the DOS can tell when a key is being held down, and repeat it — this is how the typematic action is accomplished on PC's. Since the repeat rate, and the amount of time you must hold down a key before repeating starts, is controlled by DOS and not the keyboard, it follows that this rate should be adjustable. And indeed it is, using the DOS MODE command. The KEYB and MODE commands are discussed in Chapter 9.

Since it is the operating system that determines the meaning of the scan codes, it is easy to modify this meaning so that the same keyboard can be used for different languages. This can be accomplished with the DOS KEYB command.

There is a large variety of keyboard layouts today, particularly among laptop computers. But at least for desktop machines, nearly all are identical or very similar to the layout shown in Figure 1-1, which is known as the Enhanced Keyboard Layout.

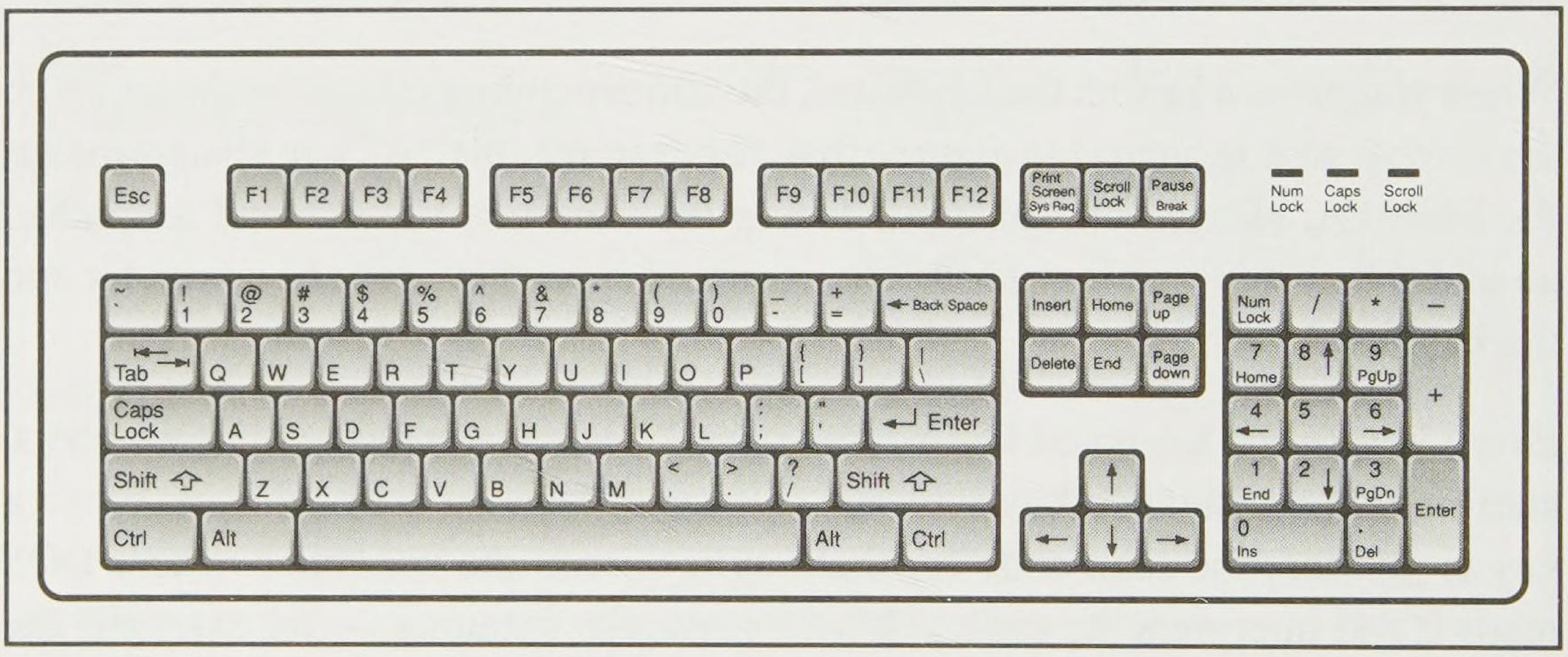

Esc
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
F11
F12
Print Screen Sys Req
Scroll Lock
Pause Break
Num Lock
Caps Lock
Scroll Lock
~ `
! 1
@ 2
# 3
$ 4
% 5
^ 6
& 7
* 8
(9
) 0
_ -
+ =
Back Space
Insert
Home
Page up
Num Lock
/
*
–
Tab
Q
W
E
R
T
Y
U
I
O
P
{ [
}]
| \
Delete
End
Page down
7 Home
8
9 PgUp
+
Caps Lock
A
S
D
F
G
H
J
K
L
: ;
" '
Enter
4
5
6
Shift
Z
X
C
V
B
N
M
< ,
> .
? /
Shift
1 End
2
3 PgDn
Enter
Ctrl
Alt
Alt
Ctrl
0 Ins
. Del

Figure 1-1 — Enhanced Keyboard Layout.

While keyboards remain the predominant method of communicating with the computer, a number of other technologies have appeared that are becoming increasingly important. The most common of these is the mouse, which is essential for running graphics programs, such as Computer Aided Design (CAD) packages, and graphic environments such as Microsoft Windows. For situations in which desk space is very limited, an alternative to the mouse is the trackball. A trackball is like an upside down mouse; the ball is on top, and instead of moving the device around the desk, you roll the ball under your palm. It is not as convenient or easy to use as a mouse, but in tight spots it is an acceptable alternative.

Disk Drives

Disk drives are used for permanent storage of information. When you save a file in your word processor, it is saved on a magnetic disk. There are three main reasons that disk drives are used: they provide "nonvolatile" storage, which means that the information is not lost when you turn off your computer; they provide the capability to store large quantities of information at low cost; and they provide a convenient way of transporting information between computers. For example, when you purchase a new software program, it comes on a magnetic disk.

Computers can accomplish only two tasks with disk drives: they can read information from the disk into memory, and they can write information onto the disk from memory. Anything else the computer does with the information stored on a disk must be done with a copy that is read into memory.

Disk capacity is measured in kilobytes and megabytes. Because electronic memory is also measured in kilobytes and megabytes, and because both disks and memory store information, people sometimes confuse the two. The essential difference between them is that disks are used for permanent storage of information, while memory is used to temporarily store information while the computer acts on it. Everything in the computer's memory is lost when you turn the power off. Everything on the disks remains.

There are two types of disks: floppy disks and hard disks. Floppy disks are inserted into disk drives on the front or side of your computer. There are two physical sizes of floppy disks: 3-1/2 inch and 5-1/4 inch. For 5-1/4 inch disks there are five different capacities supported by DOS 5.0, depending on the number of sides that can be used and the density of the magnetic particles. For 3-1/2 inch disks there are three different capacities supported. Both sides of 3-1/2 inch disks are always usable. The different sizes and capacities are shown in Table 1-2.

Table 1-2 — Floppy Disk Capacities.

Disk Size	No. of Sides	Capacity
5-1/4 inch	1	160K
5-1/4 inch	1	180K
5-1/4 inch	2	320K
5-1/4 inch	2	360K
5-1/4 inch	2	1.2M
3-1/2 inch	2	720K
3-1/2 inch	2	1.44M
3-1/2 inch	2	2.88MM

The 160K, 180K, 320K, and 360K 5-1/4 inch disks, and the 720K 3-1/2 inch disk, all have a low density of magnetic particles, and are therefore used with low density drives. The 1.2M 5-1/4 inch and 1.44M 3-1/2 inch disks both have a high density of magnetic particles and therefore require high density drives. The 2.88M 3-1/2 inch disk requires yet another type of drive, specially made to handle its very high capacity. High density disk drives can read and write both low and high density disks, but low density drives can only read and write low density disks.

Unlike floppy disks, which can be easily removed from a computer, hard disks are permanently installed. They have two main advantages over floppy disks: they can store much more information; and they can move information to and from electronic memory much faster. Recall that the computer can only operate on information that has been copied from a disk into memory. Since the computer operates at electronic speeds, the mechanical processes of reading from and writing to disks is the slowest operation the computer performs. Because hard disks are extremely fast at reading and writing, compared to floppy disks, this speed increase alone is enough to justify their existence.

Hard disks come in many sizes. The smallest is 10 megabytes, but you rarely encounter hard disks that small anymore. Sizes today range from 20 up to 650 megabytes and more.

Disk drives are referred to by letters of the alphabet. If a computer contains two floppy drives, they are probably referred to as drive A and drive B. If a computer contains a hard drive, that drive is almost always referred to as drive C. If a computer contains only one floppy drive and a hard drive, the floppy drive is generally designated to be both A and B, while the hard drive is drive C. Additional drives are referred to as D, E, F, etc. When specifying a drive, the letter is followed by a colon. Thus, you type **A:** to reference drive **A**.

Information on disk drives is stored in structures called "files." There are essentially two types of files: executable files and data files. An executable file instructs the computer to do something. A data file contains information that an executable file can manipulate. An example of an executable file is a word processor program; an example of a data file is a letter you write in your word processor.

DOS refers to files by their file names, which consist of two parts: a name and an extension, separated by a period. The name can include up to eight characters. The extension, which is optional, can contain up to three characters. The extension is commonly used to indicate the type of information stored in a file — you may use the extension LET for all of the files that contain letters you have written. Examples of file names are WP.EXE and JIMSMITH.LET.

Several extensions have special meaning to DOS, and therefore should not be used for other purposes. These special extensions are EXE, COM, BAT, and SYS. All of these refer to files that DOS can execute. In fact, DOS cannot execute a file whose name does not end with one of these extensions. All software programs — word processors, spreadsheets, database managers, etc. — are executable files with extensions of either EXE or COM. Files with BAT extensions are "batch" files, which consist of batches of one or more DOS commands that are executed in sequence. It is a simple task to create your own batch files and they are discussed in detail in Chapter 8. The fourth extension, SYS, refers to files that contain special commands used by DOS when you start your computer. These are discussed in Chapter 9.

On hard disks, where the number of files can be quite large, files are organized into tree structures called directories. Files and directories are discussed in detail in Chapter 5.

Electronic Memory

Memory refers to electronic storage locations inside the computer. Memory is measured in bytes, kilobytes, and megabytes. The computer keeps track of what is in memory, and where it is located, by assigning an "address" to each byte in the computer. The amount of memory the computer can make direct use of is determined by the CPU and the operating system. The determining factor in how much memory the computer can use is the size of the address the CPU and operating system are capable of using.

The original PC contained a CPU chip that could reference a 20-bit address. This meant the largest address the computer could use was at the one megabyte position (2^{20} is 1 megabyte). Consequently, DOS was also designed to use a 20-bit address. This limited DOS to addressing a maximum of 1 megabyte of memory. In fact, DOS was actually designed to be able to use only the first 640K of memory for programs and data — the rest was reserved for special system functions, such as video memory. At the time, 640K seemed like a ridiculously large amount of memory for a personal computer — most PCs were shipped with 128K or 256K of memory, and required expansion boards to add more than that.

The 80286 chip has a 24-bit address bus. This allows the chip to recognize up to 16 Megabytes of memory. The 80386 and 80486 chips have 32-bit address busses, which means they can address up to 4,294,967,296 bytes, or 4 gigabytes, of memory. Unfortunately, it is much easier to change the design of the computer than the design of the operating system, so even DOS 5.0 must still struggle with fitting all of its program instructions and data into the one-megabyte address space. A number of schemes have developed, and become industry standards, for helping programs exceed this limit while still using DOS as the operating system. DOS 5.0 has been designed to facilitate these schemes, which are discussed in detail in Chapter 4.

There are two types of information that can be stored in memory: instructions that the computer executes, and data that the computer operates on. Regardless of the nature of the information, it must be read into electronic memory before the computer can do anything with it. One of the interesting things about the design of computers is that the same memory is used for both programs and data. Memory is treated as one large pool, to be divided up as needed. When you run a program, the instructions for the program are read into memory, and whatever is left over is used for data. Since there is frequently not enough room in memory for all of the data a program needs, software has ways of moving information between the disk and memory as it is needed. There are ways of configuring DOS 5.0 to increase the efficiency of this process. How all of this is managed is discussed in later chapters.

DOS

As stated earlier, the CPU is the hardware component that controls and coordinates all of the other parts of the computer. But if the CPU is the brains of the computer, then the **D**isk **O**perating **S**ystem (DOS) is its mind — the knowledge, rules, and quirks that collectively define how the computer will behave all reside in DOS. DOS is the software that tells the CPU how to control and coordinate all the various tasks the computer carries out.

DOS is referred to as a "disk" operating system because it is permanently stored on disk, rather than in the computer. This is one of the design features of the IBM PC that make it so flexible — if you want a different operating system, or a new version of DOS, you simply boot the computer with the appropriate disk.

When you start your computer (a process known as "booting"), one of the first things it does is load some of the DOS programs from disk into memory. Until these programs are loaded this is all the computer knows how to do. Control of the computer is then transferred to one of these programs. From then on, everything the computer does is controlled by DOS. If you enter a command to run your word processor, it is a DOS program that interprets your request, loads the program, and tells the computer how to execute it. While the word processor is running, it makes use of a number of services that DOS provides, such as displaying things on the screen, receiving commands from the keyboard, and sending output to the printer. When the word processor is finished, control is returned to DOS, which then waits for another command from the keyboard. DOS can therefore be thought of as a master program which controls everything else the computer does.

DOS consists of a large number of programs, which fall into two broad categories. The first category consists of the programs that are essential for the computer's operation — the ones that receive keyboard input, read and write disks, etc. These programs are used by other programs; they are not invoked by typing commands in directly from the keyboard. The second category is a set of programs that can be thought of as utilities, which are provided to make your life easier by automating a lot of tasks. Examples of these are the DIR program, which displays a list of the files that are on a disk, and the MS-DOS editor, a mini-word processor that comes with DOS 5.0.

The DOS programs that are loaded into memory when the computer boots remain in memory until the computer is turned off. Other DOS programs remain on the disk, and are only loaded when you run them, just like any other software. The programs that are stored in memory are the ones that provide the essential operations necessary for your computer to function, and the utility programs that are used most often, such as the DIR program. The programs that are left on the disk are generally used less often.

The reason for not loading all of the DOS programs into memory is that they take up space. Recall that DOS can only make use of a maximum of 640K of memory. When DOS is loaded, it takes some of that memory for itself. So the more DOS programs there are in memory, the less space there is for other programs to run. DOS 5.0 is the first version of DOS that can actually load most of itself into memory that lies outside of the conventional 640K, leaving more of the 640K available for your programs and data.

DOS is not the only operating system that will run on IBM-compatible Personal Computers, but it is far and away the most common. Examples of other operating systems that can run on Personal Computers are Unix and OS/2. If a different operating system is used, then all of the commands you learn for DOS will be useless and none of your software will work, unless the operating system is specifically designed to mimic DOS. Unix is quite different from DOS; OS/2 has been designed as an evolution from DOS and therefore has some similar commands. It also includes a "DOS window" which behaves just like DOS and allows you to run DOS programs.

DOS is sometimes referred to as MS-DOS, and sometimes as PC-DOS. PC-DOS refers to the DOS that comes with Personal Computers manufactured by IBM; MS-DOS refers to Microsoft DOS, which is the DOS that comes with any other IBM-compatible Personal Computer. These two releases of DOS are essentially the same; IBM occasionally makes minor internal adjustments to fit DOS to their own hardware design, but for the purposes of this book the terms PC-DOS and MS-DOS are identical, so the word DOS is used without a prefix.

DOS was originally designed as a "command line" operating system, which means that you had to type in a command to make the computer do anything. This meant memorizing a great many commands, and keeping a good mental picture of how your files were organized on your hard disk. DOS 4.0 changed all that by introducing a feature known as the "Shell," which provides a menu-driven interface to DOS commands, to the files on your disks, and to the programs you run on your computer. DOS 5.0 has considerably enhanced the Shell. Consequently, there are now two quite different ways of accessing the many features of DOS: via the command line, and via the Shell. Each has its own place in the world of computing; both are covered in detail in the following chapters.

Chapter 2

The DOS 5.0 Shell

For most users of Personal Computers — everyone other than programmers and software developers — the single most important event in the evolution of DOS may have been the Shell that first appeared in Version 4.0. The Shell does away with the necessity of memorizing DOS commands, and remembering the details of how a hard disk is organized. It provides menu-driven screens, with online help, for running programs, executing DOS commands, and managing your disks.

The Shell has been significantly enhanced in DOS 5.0. These enhancements include:

- a new interface that is very similar to that of Windows 3.0
- less memory used by the Shell
- the ability to nest program groups several levels deep
- the ability to create multiple help screens linked with a hypertext feature
- the ability to run multiple programs and switch between them at the touch of a key.

This chapter will explore the Shell in detail, discussing all of its features and explaining how to customize it to fit your own particular needs.

The Shell can be displayed either in text mode or in graphics mode (if your computer has a graphics video card and monitor). Graphics mode uses icons, or small pictures, to represent different items in the Shell screen. Since text mode is available on all computers, the figures in this chapter will show the text mode version of these screens.

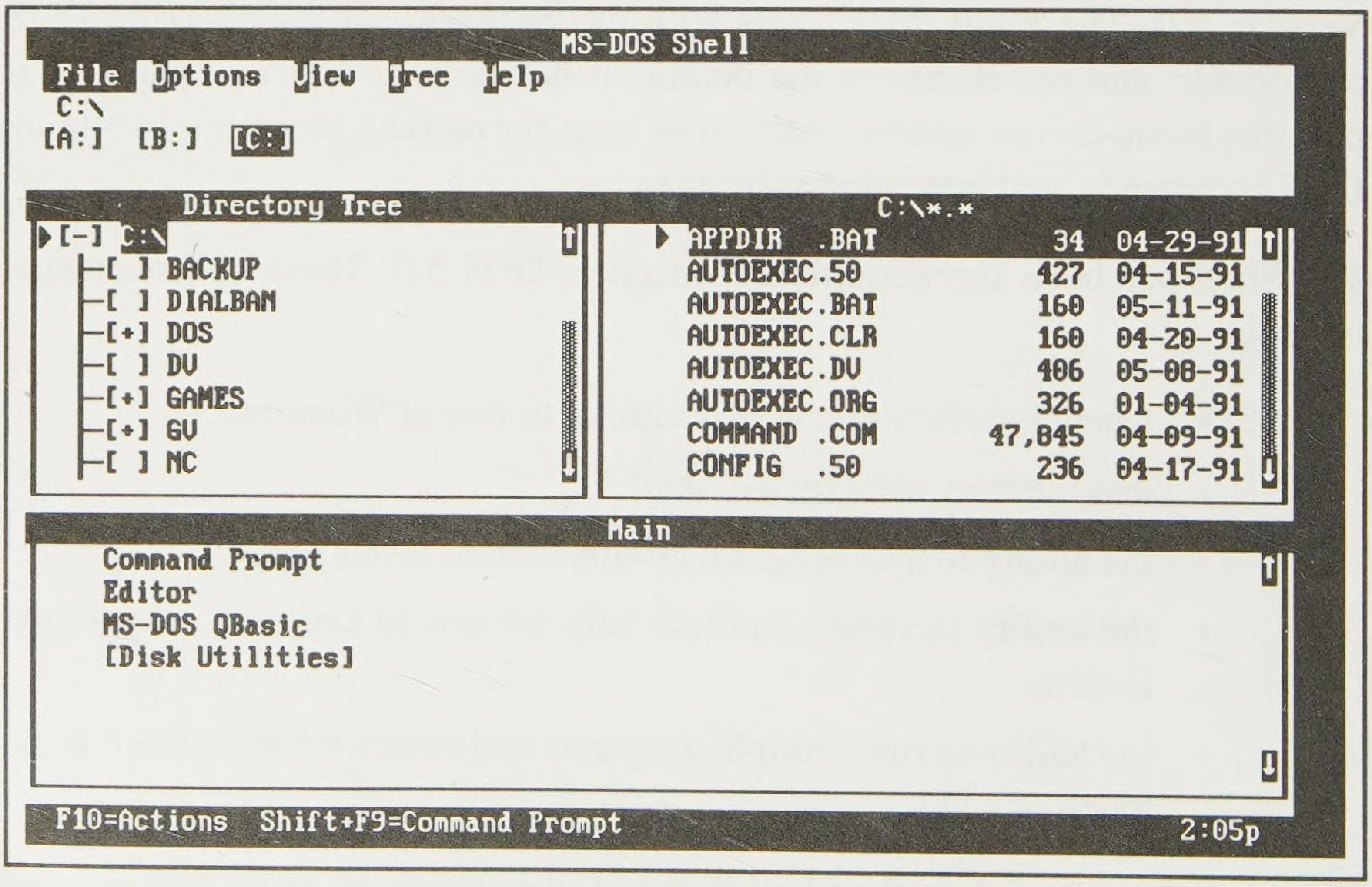
The best way to learn about the Shell is to experiment with it. As you read through this chapter, you are encouraged to take the time to try out the various features.

Overview of the Shell

If, during installation of DOS 5.0, you indicated that the Shell should come up automatically when the computer starts, then when the computer is booted, the first screen to appear will be the MS-DOS Shell screen. If the Shell does not automatically display, it can be brought up by entering the following command at the C:> prompt:

```
DOSSHELL
```

There are several distinct areas on the MS-DOS Shell screen, each of which has its own title. This screen is shown in Figure 2-1.



```
                              MS-DOS Shell
 File  Options  View  Tree  Help
 C:\
 [A:]  [B:]  [C:]

        Directory Tree                          C:\*.*
▶[-] C:\                            ▶ APPDIR   .BAT         34  04-29-91
   ├─[ ] BACKUP                       AUTOEXEC.50          427  04-15-91
   ├─[ ] DIALBAN                      AUTOEXEC.BAT         160  05-11-91
   ├─[+] DOS                          AUTOEXEC.CLR         160  04-20-91
   ├─[ ] DV                           AUTOEXEC.DV          406  05-08-91
   ├─[+] GAMES                        AUTOEXEC.ORG         326  01-04-91
   ├─[+] GV                           COMMAND  .COM     47,845  04-09-91
   ├─[ ] MC                           CONFIG   .50         236  04-17-91

                                   Main
     Command Prompt
     Editor
     MS-DOS QBasic
     [Disk Utilities]

 F10=Actions  Shift+F9=Command Prompt                                2:05p
```

Figure 2-1 — MS-DOS Shell Screen.

The names of files and directories will be different on each PC, but the areas will be the same. The MS-DOS Shell screen contains the following areas:

- The Title Bar, which displays "MS-DOS Shell."
- The Menu Bar, which is like a master control center for the Shell, provides access to a variety of features available for manipulating files, running programs, and customizing the Shell.
- The Drive Icon area, which displays the list of disk drives on the computer. The currently selected drive is highlighted.
- The Directory Tree area, which shows a graphic tree of the directory structure on the currently selected disk drive.
- The File List area, which shows the individual files in the directory that is selected in the Directory Tree area.
- The Program List area, which shows a list of programs that can be run, as well as menus that can be displayed for other program lists.
- The Status Bar, which indicates certain shortcut keys and the current time.

The screen shown in Figure 2-1 is the default Shell startup screen. When you modify the Shell, as discussed later in this chapter, the modifications are saved in a special file named DOSSHELL.INI. This file is read each time the Shell is started, so you always come back to the same configuration you left. If your screen does not appear as shown in Figure 2-1, it is because the last time the Shell was exited, the screen layout was different from the default startup screen.

When the Shell starts, it reads the entire directory structure on the disk. While it is doing this, it displays a dialog box in the middle of the screen, showing its progress. This may take a little time, but you do not have to wait for it to finish to use the Shell menus and programs. You can go ahead and select items right away, and the Shell will return to reading the directories whenever it has time.

Navigating in the Shell

At any given time, one area of the screen is the active area — the area in which you may select items. On a color monitor, the active area can be spotted quickly because the title color changes. Within the active area, there is a currently selected item — the program you are going to run, or the file you are going to copy.

A Selection Cursor indicates the currently selected item within the active area. If you have a color monitor, the Selection Cursor is indicated by a color difference in the name of the item, and a small right-pointing arrow appears to the left of the item. If you have a monochrome monitor, it is indicated just by the arrow. (With either type of monitor, the small arrow will not appear when you are in the drive list area at the top of the screen. The current drive appears in a different color, or in reverse video.) In Figure 2-1 the Selection Cursor is on the first directory (C:\) in the Directory Tree.

If you are using a mouse, the mouse pointer appears in addition to the Selection cursor. In text mode, the mouse pointer looks like a square block that appears in reverse video. In graphics mode, the mouse pointer is represented by an arrow that points toward the upper left corner of the screen. It is the Selection Cursor, not the mouse pointer, that indicates a currently active location. The mouse pointer can be anywhere on the screen, but it has no effect until you click the mouse button, which moves the Selection Cursor to where the mouse pointer is, and activates that item.

The tab and ⇧Shift-tab are used to move between areas of the screen. When the Shell starts, the Drive Icon area is active, and the current drive is selected. Pressing tab moves the Selection Cursor to the Directory Tree area, making that the active area. The currently selected item will be the first directory in the tree. Pressing tab again moves it to the File List area.

Within the active area, the arrow keys are used to move between items. If you have a mouse, then you can jump directly from one area and item to another, by positioning the mouse cursor on the new item and clicking the left mouse button.

Using Scroll Bars

There is a vertical scroll bar along the right edge of the Directory Tree, File List, and Program List areas. In Figure 2-1, the lower part of the scroll bars in the Directory Tree and File List areas appears in a hatch pattern, while the upper part appears white. On a color monitor these areas will be in different colors. The white part of the scroll bar is called the scroll box. The presence of a scroll box indicates that there is a longer list than can fit on the screen. The position of the scroll box indicates the section of the list that currently appears. For example, the scroll bar for the Directory Tree area in Figure 2-1 indicates that the top portion of the directory tree is currently displayed on the screen. The size of the scroll box indicates the percentage of the list that is displayed. In Figure 2-1, about 30 percent of the entire Directory Tree is displayed, and about 20 percent of the File List is displayed. No scroll box appears in the scroll bar for the Program List area, indicating that the entire list appears on the screen.

When a scroll box is shown, a mouse can be used to scroll up and down through the list. There are three ways to do this:

1. Position the pointer on the scroll box, and hold down the left mouse button. The scroll box can now be dragged up and down with the mouse. As it is dragged, the display will change to show the different sections of the list.
2. Position the mouse on the up or down arrow. Briefly clicking the left button once will scroll the list up or down by one item. Holding the button down will scroll the list continuously, one line at a time.
3. Position the pointer anywhere on the scroll bar above or below the scroll box. Briefly clicking the left button will scroll the list one full screen toward the mouse pointer. Holding the button down will scroll the list continuously, one screen at a time, until the scroll box meets the pointer.

There are also three ways of using the keyboard for scrolling:

1. ↑ and ↓ scroll the Selection cursor through the list one item at a time.
2. PgUp and PgDn scroll through the list one full screen at a time.

3. Pressing the first letter of an item in a list jumps to that item. For example, pressing R when the Directory Tree is the active area will immediately jump to the first directory name that begins with an R. Pressing R again jumps to the next one that begins with R, and so on. If there are no items in the list that begin with the letter pressed, the cursor will not move.

Changing the View

When the Shell starts, the display looks similar to Figure 2-1. This screen provides a particular view of all the information available within the Shell. However, there may be times when this is not the most convenient view. For example, to work on file management it would be helpful to see more of the Directory Tree and File List than this view affords. There are a total of five different views available in the Shell. They can be selected from the View menu shown in Figure 2-2.

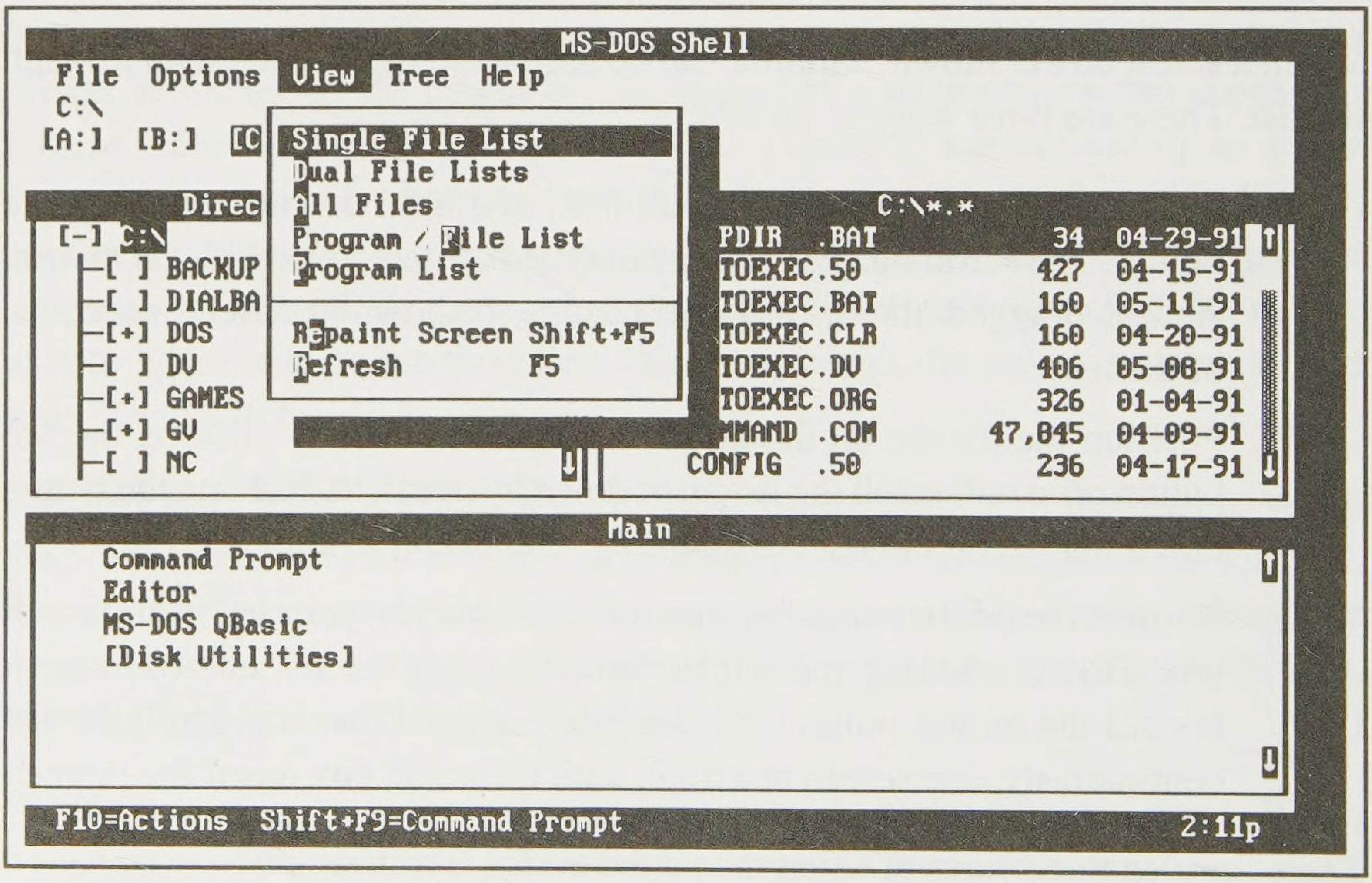

MS-DOS Shell
File Options View Tree Help
C:\
[A:] [B:] [C
Single File List
Dual File Lists
All Files
Program / File List
Program List
Repaint Screen Shift+F5
Refresh F5
Direc
[-] C:\
[ ] BACKUP
[ ] DIALBA
[+] DOS
[ ] DV
[+] GAMES
[+] GV
[ ] NC
C:\*.*
PDIR .BAT 34 04-29-91
TOEXEC.50 427 04-15-91
TOEXEC.BAT 160 05-11-91
TOEXEC.CLR 160 04-20-91
TOEXEC.DV 406 05-08-91
TOEXEC.ORG 326 01-04-91
MMAND .COM 47,845 04-09-91
CONFIG .50 236 04-17-91
Main
Command Prompt
Editor
MS-DOS QBasic
[Disk Utilities]
F10=Actions Shift+F9=Command Prompt
2:11p

Figure 2-2 — The Shell View Menu.

To select a new view, simply highlight your choice on the View menu and press Enter. The different views will be discussed in detail later in this chapter, but the following is a brief description of what each one offers.

The **Single File List** view uses the entire screen to display the Directory Tree and File List areas — the other areas of the Shell are hidden. **Dual File Lists** splits the screen horizontally, displaying two Directory Tree and File List areas. These areas can show the directory and file information for two different disks, or for two different directories on the same disk. **All Files** lists all of the files on the current drive, along with technical information about the disk and the files. The Directory Tree is not displayed. The **Program/File Lists** option is the default view shown in Figure 2-1. **Program List** uses the entire screen to show the Program List.

Under certain circumstances the Shell screen may have some residue from another program left on it. For example, if you are running a TSR (Terminate and Stay Resident) program, and call it up over the Shell, the Shell screen may not be restored properly when you exit the TSR. If this happens, choosing Repaint Screen from the View menu will restore the Shell screen. You may also repaint the screen by pressing ⇧Shift-F5.

There are circumstances under which the Directory Tree and/or File List may not be accurate. This happens when a program you run adds, deletes, or changes a directory or file. The Directory Tree and File List are read when the Shell starts, but they are not normally updated until the Shell is permanently exited and restarted. Therefore, when a program running from within the Shell makes a change, that change will not be reflected in the displayed Directory Tree or File List. The Refresh option on the View menu updates the display. Pressing F5 has the same effect. This forces the Shell to re-read the disk, and update the display accordingly.

Selecting Disks

Any view that includes a Directory Tree or File List area also shows the Drive Icon area. To select a different drive, press tab until the Drive Icon area is active (this may be a little difficult to see on some monochrome screens). Then use the arrow keys to highlight the drive you want to select. Press Enter← after the drive is highlighted. The Shell will read the disk in that drive and display its directory tree and file list.

If you are displaying a single file list on the screen and select a different drive, the information for the first drive will be lost. It can take a while to read the complete directory tree and file list from a large hard disk, so it is often advisable to choose the Dual File List view before switching drives. This keeps the original directory tree and file list in memory while the second drive is read and displayed.

Exiting the Shell

You may exit the Shell to work at the command prompt in either of two ways: permanently or temporarily. There are several reasons for wanting to work at the command prompt: you may find it faster to type in specific commands than to navigate through menus; you may want to free up the memory that the Shell occupies, to provide more memory for a program; or you may want to use DOS commands that are not available within the Shell. (If there is a DOS command you would like to have available in the Shell, you can always add it yourself, using techniques described later in this chapter).

When you exit the Shell temporarily, part of it remains in memory, allowing you to return to it very quickly. This is convenient if you have just a few tasks to do at the command prompt and want to come back to the Shell. The disadvantage is that leaving part of the Shell in memory uses up about 5K of memory that would otherwise be available for programs.

There are two ways to temporarily exit the Shell: pressing ⇧Shift-F9; or choosing the Command Prompt option on the Program List menu. After a moment the Shell will disappear and the traditional command prompt will be displayed. To return to the Shell from the command prompt, enter the EXIT command.

It is worth understanding how this works. The principle is the same for any program that has the ability to temporarily exit to the command prompt. There is a DOS program named COMMAND.COM that is loaded into the computer's memory at boot time. COMMAND.COM interprets the commands that are sent to the computer. This includes commands that come from programs, so when a program is running, COMMAND.COM is needed to manage the program's execution. The Shell is a program, so it requires the attention of COMMAND.COM. When you exit the Shell temporarily, the Shell is still in memory and therefore continues to require COMMAND.COM's attention. But the reason for exiting to the command prompt is to type in more commands. Since the copy of COMMAND.COM that was loaded when the computer was started is tied up monitoring the Shell, another copy of COMMAND.COM must be loaded into memory. This second copy of COMMAND.COM is unaware that the Shell is in memory, and is therefore free to monitor the commands typed in at the keyboard. COMMAND.COM takes up about 2.5K of memory, and the Shell takes up about 2.1K. So when you temporarily exit the Shell, the total memory required is about 4.6K. Because of minor inefficiencies in the way in which these various programs load into memory, there are a few extra tenths of a K used up, and almost exactly 5K is lost to the Shell.

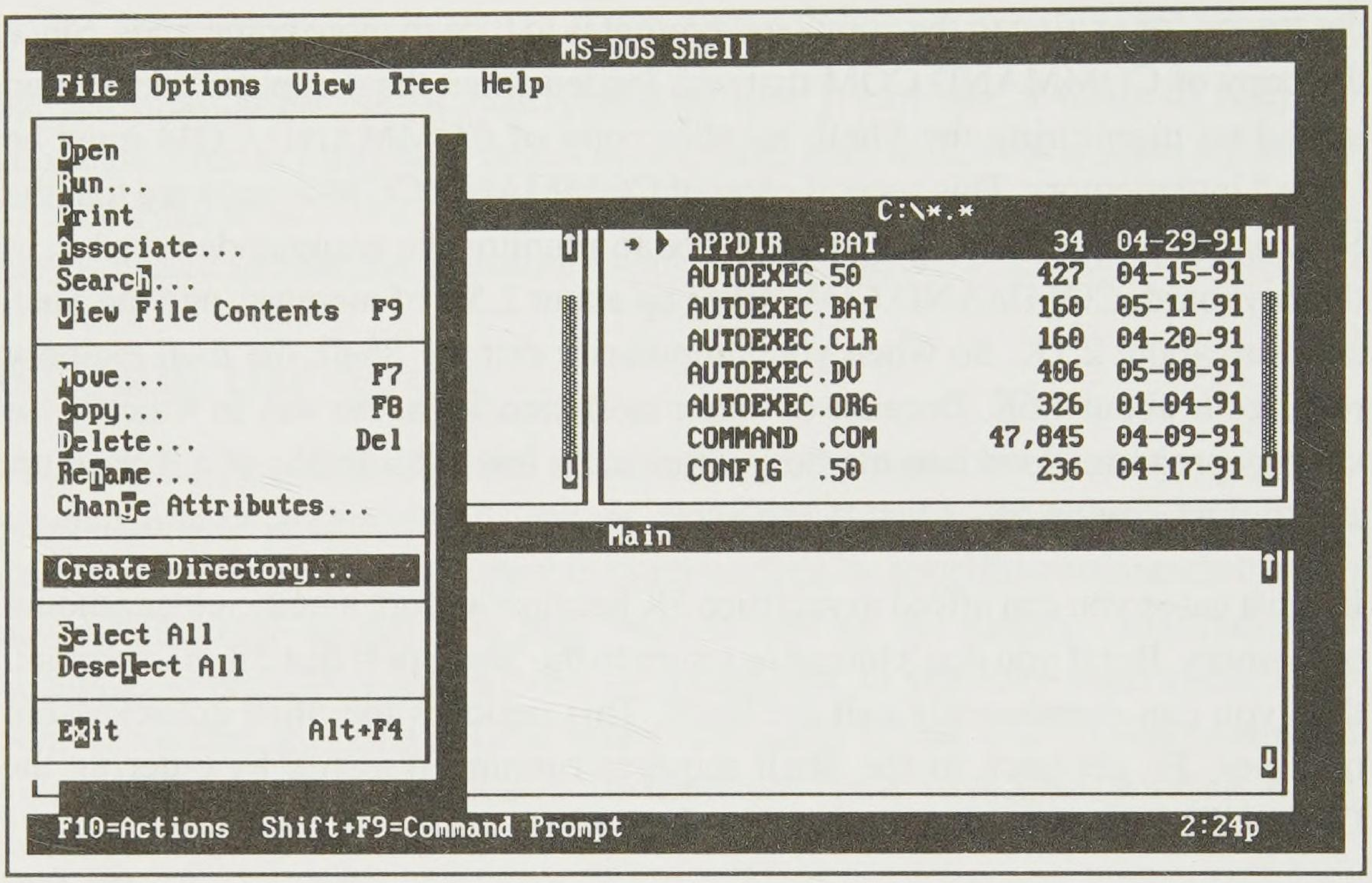
In most cases you can afford to sacrifice 5K because it is not a tremendous amount of memory. But if you don't intend to return to the Shell, or if that 5K is important, then you can permanently exit the Shell. This removes the Shell entirely from memory. To get back to the Shell requires running it again, by entering the DOSSHELL command at the command prompt.

There are three ways to exit the Shell permanently: pressing F3, pressing alt-F4, or choosing Exit from the File menu.

If you have the Task Swapper enabled, and there are programs running under it, you will not be able to permanently exit the Shell. If you try, a dialog box will tell you that you cannot quit the Shell without first quitting all of the programs that are running. The names of all programs running in the Task Swapper are shown in the Active Task List.

Using Shell Menus and Dialog Boxes

The top of the Shell screen always displays the Menu Bar. The menu bar is activated and deactivated by pressing either alt or F10. Each of the options within the menu bar has a pull-down menu associated with it. The Shell screen with the File menu pulled down is shown in Figure 2-3. Note that the File Menu options are dependent on which part of the Shell screen is active, and that the following examples assume either the Drive area, the Directory tree area, or the File List area is active.


MS-DOS Shell
File Options View Tree Help
Open
Run...
Print
Associate...
Search...
View File Contents F9
Move... F7
Copy... F8
Delete... Del
Rename...
Change Attributes...
Create Directory...
Select All
Deselect All
Exit Alt+F4
C:\*.*
Main
F10=Actions Shift+F9=Command Prompt
2:24p

```
APPDIR  .BAT         34  04-29-91
AUTOEXEC.50         427  04-15-91
AUTOEXEC.BAT        160  05-11-91
AUTOEXEC.CLR        160  04-20-91
AUTOEXEC.DV         406  05-08-91
AUTOEXEC.ORG        326  01-04-91
COMMAND .COM     47,845  04-09-91
CONFIG  .50         236  04-17-91
```

Figure 2-3 — File Menu.

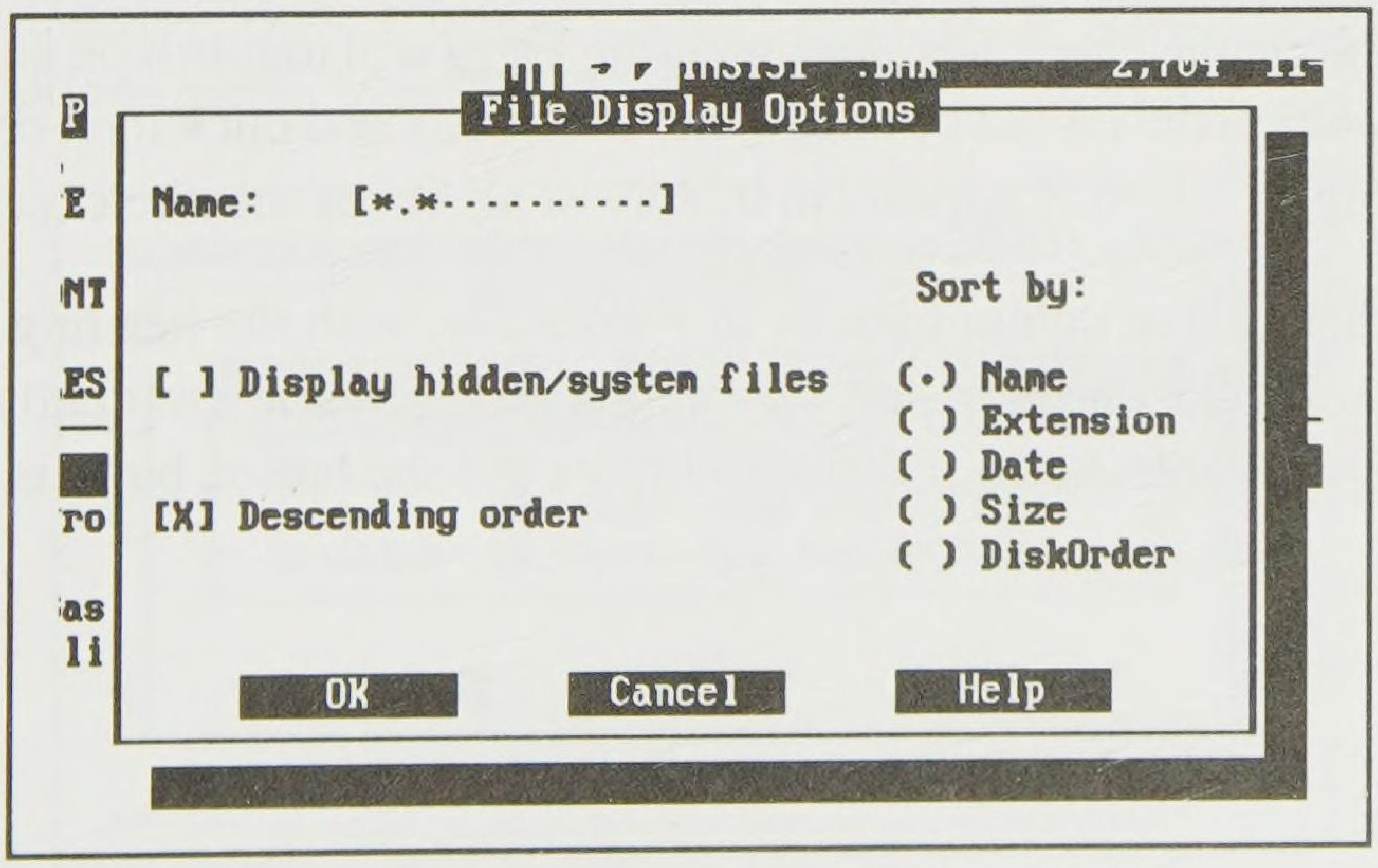
There are several variations on how menu items may appear on a menu:

1. Some of the menu options may be followed by an ellipsis (…). This means that a dialog box will appear when the option is selected. The dialog box will prompt you for some type of information. In Figure 2-3, the Run, Associate, and Search options are a few examples of menu options that bring up dialog boxes.
2. Some of the options on a menu may not be available. This is dependent on the screen context when the menu is pulled down. If an option is currently unavailable, it will be dimmed or invisible. Attempting to select it will have no effect.
3. Some menu options are switches — they turn a feature on or off. When one of these menu options is turned on, a diamond appears next to it.

There are three ways to select items on a menu:

1. You may use the arrow keys to move the highlight from one option to the next. From the top of the menu, the ↑ causes the highlight to jump to the last menu item. From the bottom of the menu, the ↓ causes the highlight to jump to the first menu item. When you have highlighted a menu option in this way, press Enter← to select it.
2. Each menu item has one letter highlighted. This letter is the "hot key" for that item and pressing it causes the highlight to jump immediately to that item and select it.
3. If you are using a mouse, you may position the mouse cursor on a menu item. Clicking once on the item will cause the highlight bar to jump to it and execute it.

When the Shell requires some input from you in order to carry out a task, it uses a dialog box. There are five devices used within dialog boxes to capture your input: command buttons, check boxes, list boxes, option buttons, and text boxes. A dialog box always has command buttons. In addition, it may also have any combination of the other four devices. For example, the File Display Options choice on the Options menu is used to select the particular files that will appear in the File List area (this is discussed in more detail later). When you choose this option, the Shell must know several things about the files you wish to display, so it shows you the dialog box in Figure 2-4.



```
File Display Options

Name:     [*.*..........]

                                         Sort by:

[ ] Display hidden/system files        (•) Name
                                       ( ) Extension
                                       ( ) Date
[X] Descending order                   ( ) Size
                                       ( ) DiskOrder
```

Figure 2-4 — Dialog Box for File Display Options.

This dialog box contains a text box labeled **Name:** This is where you specify the file name criteria for the File List area. **Name:** initially contains the characters *.* . There are two ways to edit the text in a text box:

- Typing any alphabetic or numeric key, or pressing ←BkSp will delete whatever was originally in the text box.
- Pressing → or ← leaves what is currently on the line and allows you to modify it or add to it. The ← will shift the cursor to the beginning of the text box but leave the existing text intact. The → will put the cursor at the end of the existing text. Anything you type after the arrow keys will be added to the existing text.

Text is always entered in Insert mode — if you position the cursor in the middle of existing text, whatever you type will not overwrite what is already there, but will be inserted into the existing text. delete can be used to delete individual characters in a text box. To delete a block of text, position the cursor at the beginning of the block you want to delete, then hold down ⇧Shift and move the cursor to the end of the block. This will highlight the block. Now, delete will delete the entire block.

There are two check boxes in Figure 2-4, "Display hidden/system files" and "Descending order." A check box is either checked (meaning yes, or on) or blank (meaning no, or off). In Figure 2-4, "Descending order" is checked.

There are two ways to toggle a check box on and off. From the keyboard, tab is used to highlight the check box, then pressing space will toggle it on and off. With a mouse, position the mouse pointer in the check box and click it on or off. When the check box is on, an X appears in it. Any or all boxes may be checked.

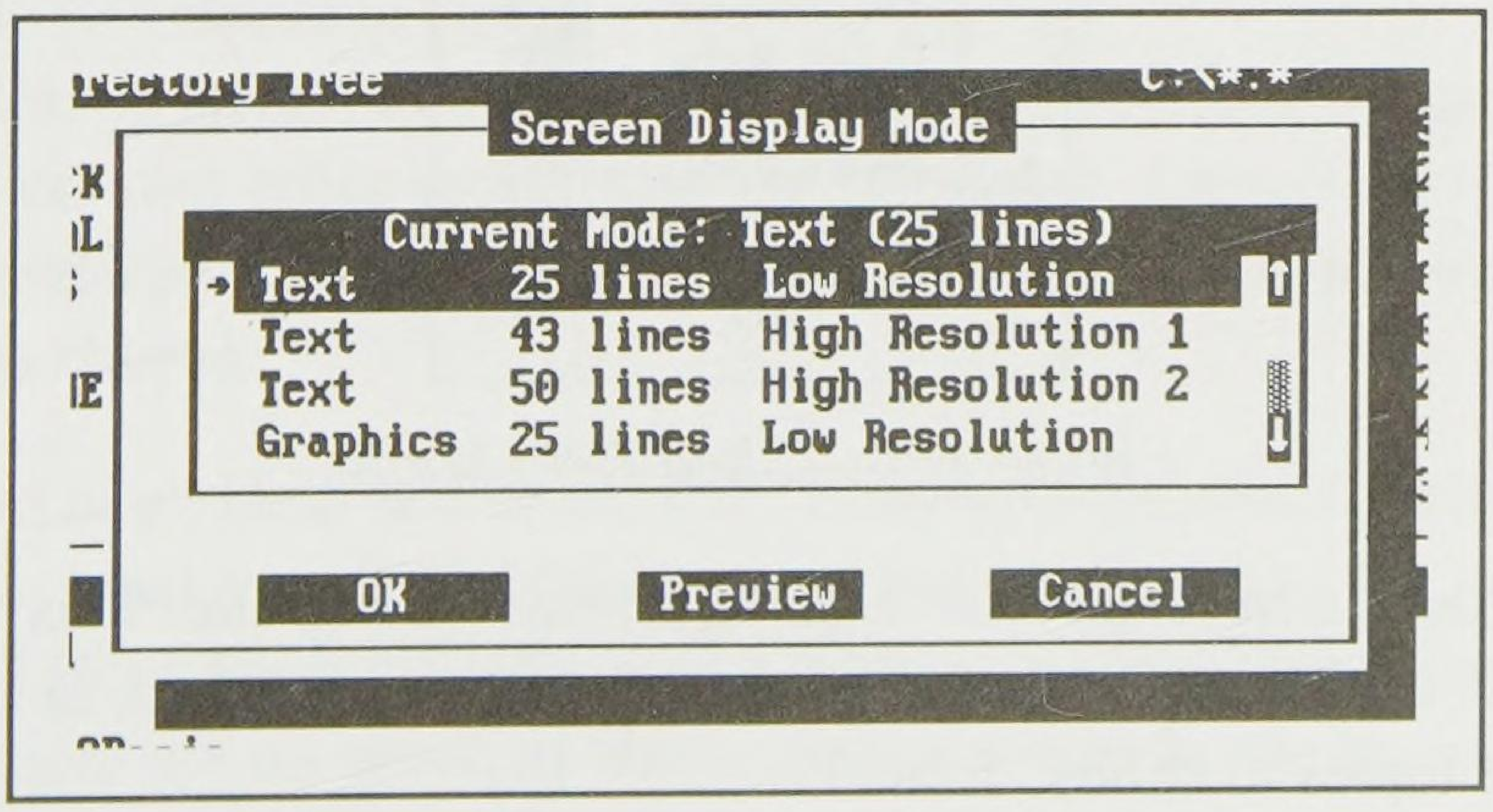
There is a list of five option buttons in Figure 2-4, with the heading "Sort by:" above them. Unlike check boxes, option buttons in lists are mutually exclusive — you may only select one of them. In Figure 2-4 the option buttons select one of several possible options for sorting the File List. There are two ways to select option buttons:

- From the keyboard, tab is used to position the cursor in the list area. The arrow keys or space may then be used to move through the list. As the cursor moves, the option it is on is highlighted.
- With the mouse, simply position the mouse pointer on the option button you want and click on it.

When an option button is selected, a dot appears in the parentheses next to it.

There are three command buttons at the bottom of the dialog box: OK, Cancel, and Help. Command buttons are used to execute actions. The **OK** button is used to indicate acceptance of the settings in the dialog box, and to execute the process that called the dialog box. The **Cancel** button is used to cancel the process. The **Help** button is used to bring up context-sensitive help that provides information about the dialog box being displayed. Another command button that appears in some dialog boxes is the "Advanced…" button. The ellipsis (…) indicates that executing this button will bring up a dialog box containing advanced features.

List boxes are the only dialog box device not represented in Figure 2-4. A list box is used to choose from among a list of items. List boxes are used where the dialog box is too small to show the complete list. They contain a scroll bar that allows you to scroll through the list. Like the option buttons, only one item in a List box can be selected at any given time. An example of a List box is seen in the Screen Display Mode box that appears when the Display choice is selected from the Options menu. This dialog box is shown in Figure 2-5.



```
      Current Mode: Text (25 lines)
→ Text       25 lines   Low Resolution
  Text       43 lines   High Resolution 1
  Text       50 lines   High Resolution 2
  Graphics   25 lines   Low Resolution
```

Figure 2-5 — Screen Display Mode Dialog Box.

Activating the Menu Bar

The alt and F10 keys are used to toggle the Menu Bar on and off. Pressing alt (or F10) will highlight the Menu Bar. Pressing it again will turn off the highlight. When it is on, only the Menu Bar is active. When it is off, the area that was active before turning on the Menu Bar is again active.

SHORTCUT

When the Menu Bar is first activated, the File option is highlighted. If you want one of the other options, you can either press the first letter of the option, or use the arrow keys to move to it. This is a two-step process: first the menu bar is activated, then the option is selected. A shortcut is to hold down alt and, before releasing it, press the first letter of one of the Menu Bar options. This takes you directly to the menu for that option. There are many functions available from the Menu Bar menus and each is discussed in the appropriate section below.

Getting Online Help

There is extensive online Help available within the Shell and there are three ways to get to this Help: choosing the Help option on the Menu Bar; pressing F1; or selecting the Help option in a dialog box. The first option is like opening a book to its table of contents; the pull-down Help menu shown in Figure 2-6 is displayed.

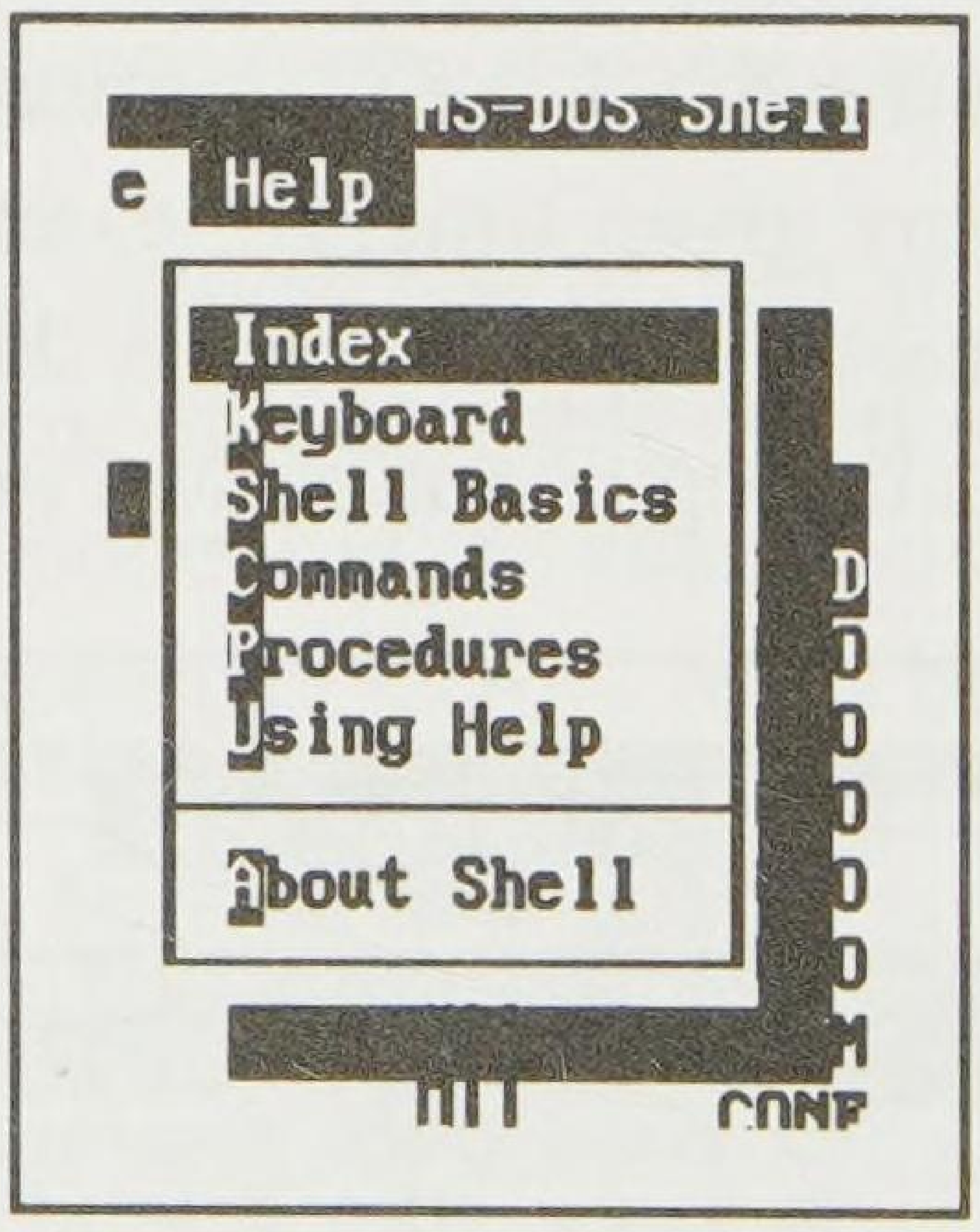
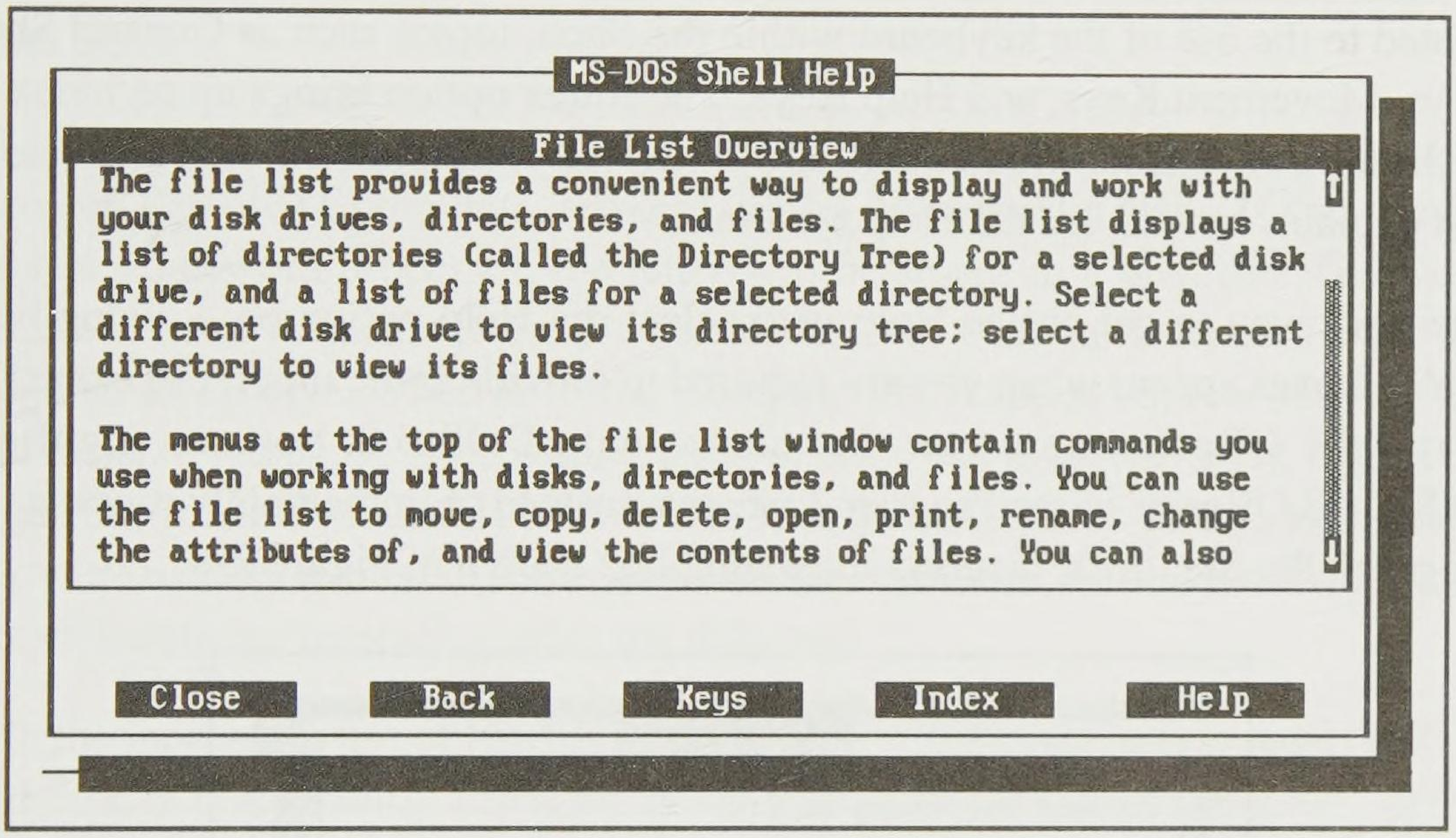


Figure 2-6 — Shell Help Menu.

From here, you can highlight the subject you want to explore. This will bring up a Help screen that has information on the selected subject and, in most cases, a further list of topics to choose from.

The second option, F1, displays a context-sensitive Help screen, which means that the Help screen that appears pertains to the area of the screen that is currently active. For example, if the File List area is highlighted, pressing F1 brings up the Help Screen shown in Figure 2-7.



```
The file list provides a convenient way to display and work with
your disk drives, directories, and files. The file list displays a
list of directories (called the Directory Tree) for a selected disk
drive, and a list of files for a selected directory. Select a
different disk drive to view its directory tree; select a different
directory to view its files.

The menus at the top of the file list window contain commands you
use when working with disks, directories, and files. You can use
the file list to move, copy, delete, open, print, rename, change
the attributes of, and view the contents of files. You can also
```

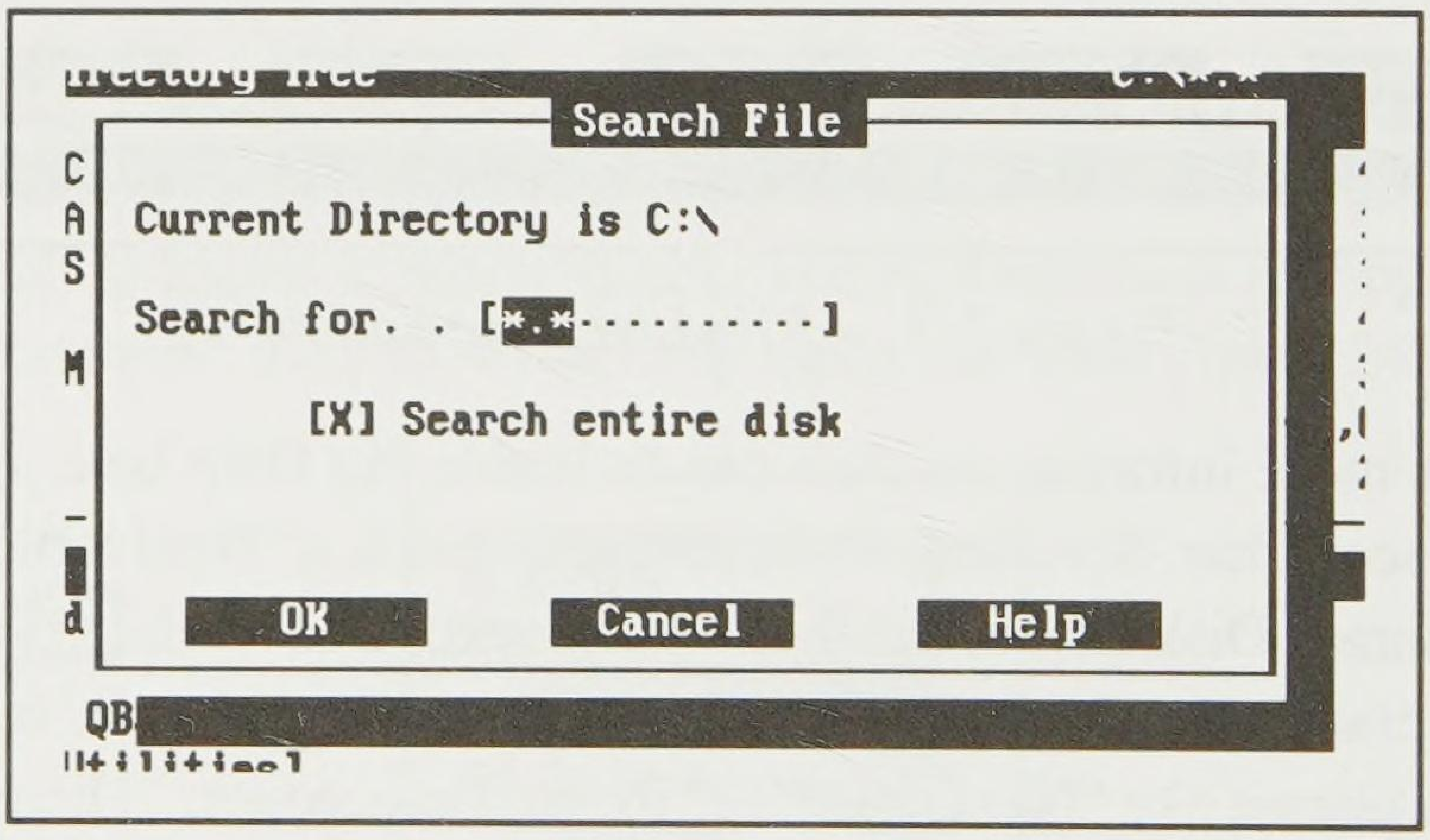
Figure 2-7 — File List Help Screen.

Since there is more information than can fit inside the Help box, a scroll box is shown in the scroll bar. Scrolling down reveals there is a "Next topic" about Disk Drives. The phrase Disk Drives can then be selected, either with tab or the mouse. When a selectable word or phrase appears on a Help screen, it is generally preceded by a phrase like "Next topic" or "Related procedure." There is additional help available for selectable topics, simply by selecting them and pressing Enter←┘, or double clicking on them with the mouse pointer. These additional help screens may in turn contain other topics that are selectable. This feature is known as "hypertext" and it permits you to move through a series of help screens in whatever sequence you choose.

At the bottom of all Help screen are five choices: Close, Back, Keys, Index, and Help. **Close** is used to end the Help session and return you to the point at which you requested Help. **Back** is used to move back to previous Help screens. Selecting Back when there are no previous Help screens results in the message "There is no previous help to go back to."

The last three options at the bottom of Help screens provide access to sections of the Help facility that do not pertain directly to areas of the Shell screen. These parallel some of the options on the Help menu, but by choosing them from a context-sensitive Help screen, you have the ability to use the Back option to return to the context-sensitive Help. The **Keys** option brings up a list of several topics related to the use of the keyboard within the Shell, topics such as General Shell Keys, Movement Keys, and Help Keys. The **Index** option brings up an index of all the topics available within the Help system. The **Help** option brings up a screen that explains how to use the Help system.

The third way to get online Help is to select the Help option on a dialog box. Dialog boxes appear when you are required to provide some input. For example, to run the QBasic interpreter that comes with DOS 5.0, you can highlight "MS-DOS QBasic" in the Program List area. Instead of immediately running the program, the Shell first displays the dialog box shown in Figure 2-8.



```
Current Directory is C:\

Search for. . [*.*..........]

      [X] Search entire disk
```

Figure 2-8 — Dialog Box for Search File Option.

This box prompts you for the name of a file to locate on the disk. If you are not sure how this box works, or what to enter, you can select the Help option at the bottom of the screen. This brings up a screen that describes the specific options available. You can achieve the same result by pressing F1 when the dialog box is displayed.

Managing Directories in the Shell

The commands on the File menu are used for managing directories and files. Although directories and files are discussed in detail in Chapter 5, a brief introduction is necessary here in order to understand the file management features of the Shell.

All of the software programs used on Personal Computers, and all of the data stored on Personal Computers, are kept in files on magnetic disks. If you write a letter in your word processor, the letter will be stored in a disk file. The word processor itself consists of one or more files that contain instructions for the computer to execute. When Personal Computers first appeared, few of them had hard disks. Organizing and keeping track of files was relatively easy, because a single disk generally had only a handful of files on it. All of the files on a particular disk were usually related in some way, and the label stuck to the front of the disk was sufficient for indicating what the disk held.

With the increasing use of hard disks, however, things changed. To begin with, a hard disk is permanently installed inside your computer, so there is no way to attach a physical label to it. Furthermore, hard disks can contain more than a thousand files, and the files will not all be related. Some will be for a word processor, some for a spreadsheet, some for the DOS programs, and so on. It therefore became necessary to provide some means of managing all this.

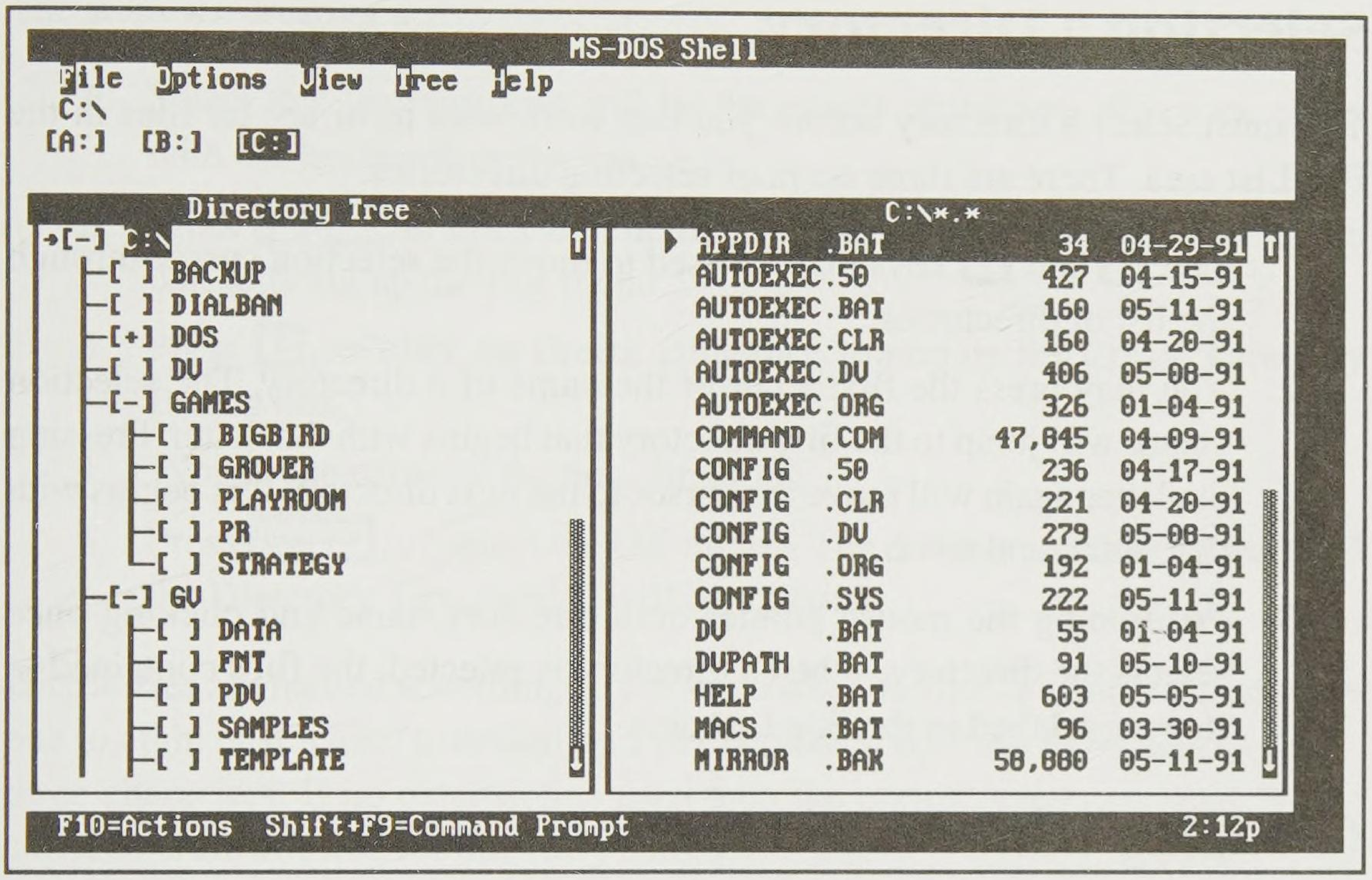
DOS Version 2.0 introduced the ability to group the files on a disk into directories. A directory can contain files and subdirectories. The subdirectories in turn can contain files and subdirectories of their own. This creates a branching structure of directories. You give the directories their names, and you decide what files will go into them, so you can create whatever file organization works best for you. Directories may be created on any kind of disk, floppy or hard, regardless of their size.

In order to manage a directory structure, DOS must be able to locate it and make sense of it. Before a disk can be used with DOS, the disk must be formatted. During the formatting process DOS creates something known as the "root directory" on the disk. The root directory always occupies a particular location on the disk so DOS can always find it. The size of the root directory depends on the type of disk; with a 360K floppy disk, the root directory can contain a total of 112 files and subdirectories. Most hard disks have root directories that contain 512 files and subdirectories. Root directories are always named by the letter of the drive, followed by a colon and a backslash. For example, C:\ is the root directory of drive C.

A directory is really just a kind of file — it stores the names and locations of the files that belong to it, and the names and locations of its subdirectories. The root directory is the only one that has a limit placed on how many files and subdirectories it can contain. There is no limit to the number of files and subdirectories that can be stored in any other directory. This provides a very flexible structure for organizing the files on a hard disk.

The terms directory and subdirectory are somewhat interchangeable: a subdirectory can be thought of as the directory of the files and subdirectories that it contains, and as the subdirectory of its parent directory. The root directory cannot be a subdirectory because it has no parent.

The Directory Tree area of the Shell screen shows a graphical representation of the directory structure. Figure 2-9 shows the Shell Single File List view of a typical hard disk.



```
                                 MS-DOS Shell
 File  Options  View  Tree  Help
 C:\
 [A:]  [B:]  [C:]

          Directory Tree                              C:\*.*
→[-] C:\                           ▶ APPDIR   .BAT        34  04-29-91
  ├[ ] BACKUP                        AUTOEXEC.50         427  04-15-91
  ├[ ] DIALBAN                       AUTOEXEC.BAT        160  05-11-91
  ├[+] DOS                           AUTOEXEC.CLR        160  04-20-91
  ├[ ] DV                            AUTOEXEC.DV         406  05-08-91
  ├[-] GAMES                         AUTOEXEC.ORG        326  01-04-91
  │  ├[ ] BIGBIRD                    COMMAND .COM     47,845  04-09-91
  │  ├[ ] GROVER                     CONFIG  .50         236  04-17-91
  │  ├[ ] PLAYROOM                   CONFIG  .CLR        221  04-20-91
  │  ├[ ] PR                         CONFIG  .DV         279  05-08-91
  │  └[ ] STRATEGY                   CONFIG  .ORG        192  01-04-91
  ├[-] GV                            CONFIG  .SYS        222  05-11-91
  │  ├[ ] DATA                       DV      .BAT         55  01-04-91
  │  ├[ ] FNT                        DVPATH  .BAT         91  05-10-91
  │  ├[ ] PDV                        HELP    .BAT        603  05-05-91
  │  ├[ ] SAMPLES                    MACS    .BAT         96  03-30-91
  │  ├[ ] TEMPLATE                   MIRROR  .BAK     58,880  05-11-91

 F10=Actions  Shift+F9=Command Prompt                                2:12p
```

Figure 2-9 — Single File List View of the Shell.

The tree structure of the directories can be seen here: C:\ is the root directory. It has numerous subdirectories: BACKUP, DIALBAN, DOS, etc. Some of these directories also have subdirectories. For example, the GAMES directory has four subdirectories: BIGBIRD, GROVER, PR, and STRAGEGY. The GV directory also has several subdirectories.

When the Shell is started, it returns to the last view that was displayed. But if the Directory Tree is part of that view, it always displays the root directory and one level of subdirectories. This is the view shown in Figure 2-1.

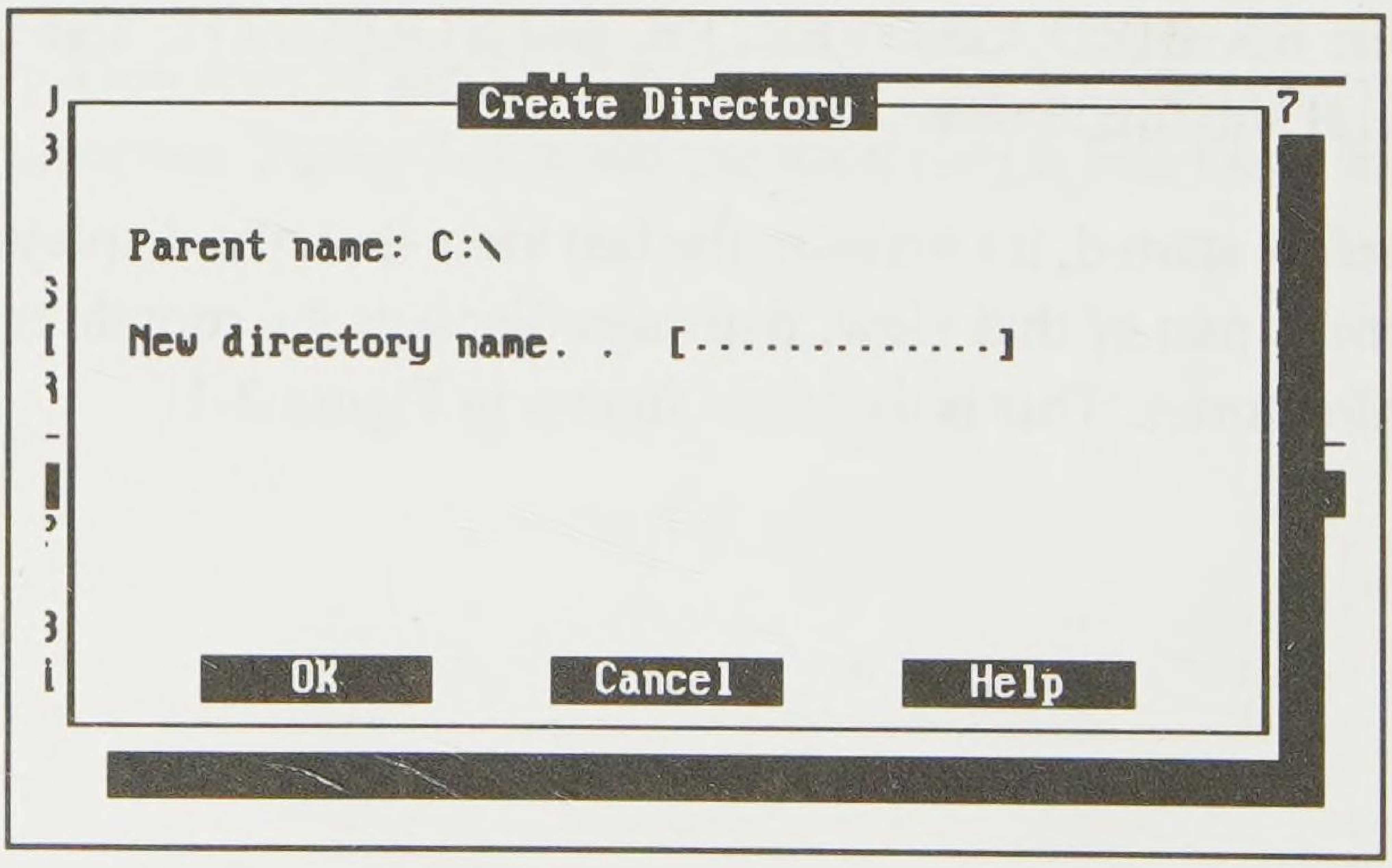
Selecting a Directory

You must select a directory before you can work with it, or see its files in the File List area. There are three ways of selecting directories:

1. The ↑ and ↓ keys may be used to move the selection cursor through the list of directories.
2. You may press the first letter of the name of a directory. The selection cursor will jump to the first directory that begins with that letter. Pressing the letter again will move the cursor to the next directory that begins with that letter, and so on.
3. Positioning the mouse pointer on a directory name and clicking once selects the directory. When a directory is selected, the files contained in it are displayed in the File List area.

Creating Directories

The commands in the File menu are used for managing files and directories. When the Directory Tree is the active area, the Create Directory option on the File menu can be used to create a new subdirectory. The Create Directory option displays the dialog box shown in Figure 2-10.



```
Parent name: C:\

New directory name. .  [.............]
```

Figure 2-10 — Create Directory Dialog Box.

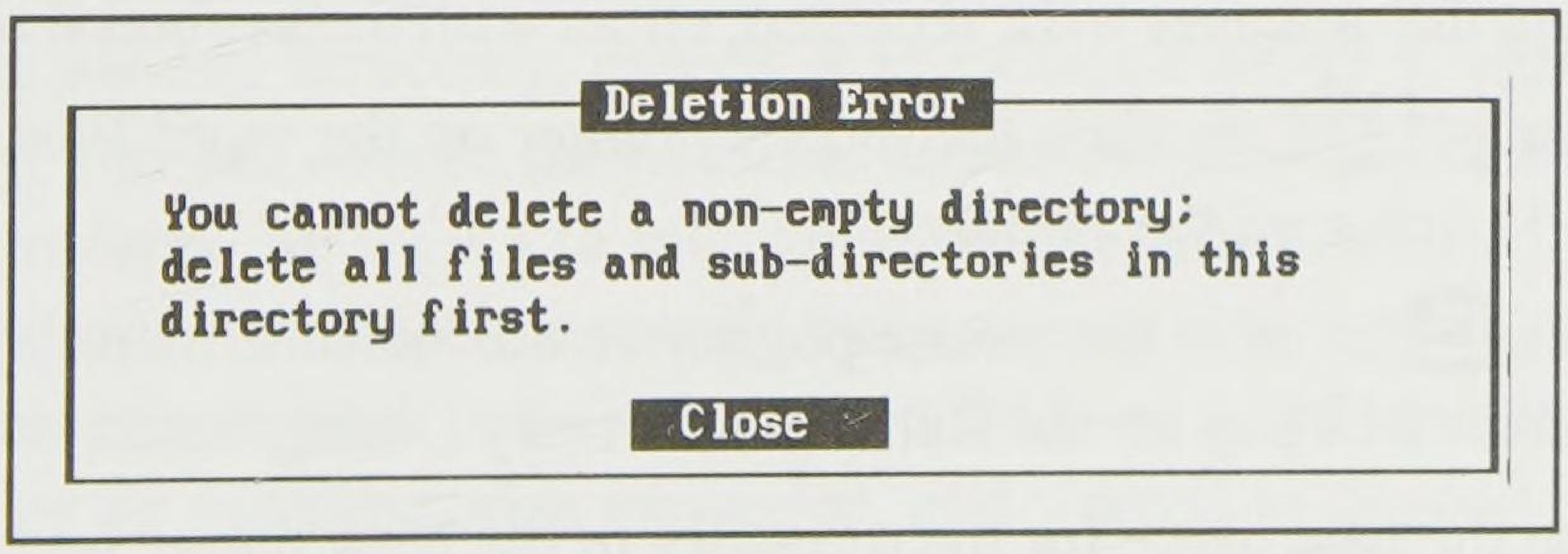
The steps for creating a new directory are:

1. Select the directory that will be the parent of the new directory, either with the keyboard or the mouse.
2. Press alt-F, or click the mouse pointer on the word File in the menu bar, to bring up the File menu.
3. Press E, or click on Create Directory, to pop up the Create Directory Dialog box.
4. Type in the name of the new directory.
5. Press Enter←, or select the OK button. The directory will be created and the Directory Tree display will be updated.

Directories are named according to the same rules as files: a name consisting of one to eight characters, followed by a period and an optional extension of one to three characters. If no extension is used then the period is also optional. While extensions are allowed for directory names, the industry standard is to leave them off. Directory names with extensions are virtually never seen.

Deleting Directories

The Delete option on the File menu is used for deleting directories. Either the Directory Tree or Drive Icon area must be the active area to delete a directory; if some other area is active, then the Delete option deletes the currently selected item in that area, rather than the currently selected directory. DOS does not allow you to delete a directory if it contains files or subdirectories. Attempting to delete a directory that is not empty will display the dialog box shown in Figure 2-11.



```
Deletion Error

You cannot delete a non-empty directory;
delete all files and sub-directories in this
directory first.
```

Figure 2-11 — Error Message when Attempting to Delete Non-empty Directory.

Note that the Delete option on the File menu is disabled if the root directory is selected, since DOS depends on the presence of the root directory for its disk management functions.

The steps for deleting a directory are:

1. Select the directory to be deleted, either with the keyboard or the mouse.
2. Delete any files in the directory, and make sure any subdirectories have been deleted.
3. Press alt-F, or click the mouse pointer on the word File in the menu bar, to bring up the File menu.
4. Press D, or delete, or click the mouse pointer on the Delete menu option. Any one of these will pop up the Delete Directory Confirmation dialog box, unless you have turned off the Confirm On Delete option (discussed later in this chapter in Managing the Shell Environment). If Confirm On Delete is off, the directory will be deleted without showing the dialog box.
5. If the Delete Directory Confirmation dialog box appears, press Enter↵, or select the Yes button to delete the directory and update the display.

Renaming Directories

Directories may be renamed using the Rename option on the File menu. As with the Delete option, either the Directory Tree or Drive Icon area must be active to rename a directory. Selecting Rename pops up a dialog box that displays the current name, and provides a text box for entering a new name. The steps for renaming a directory are:

1. Select the directory to be renamed, either with the keyboard or the mouse.
2. Press alt-F, or click the mouse pointer on the word File in the menu bar, to bring up the File menu.
3. Press N, or click the mouse pointer on the Rename menu option. Either of these will pop up the Rename Directory Dialog box.
4. Type the new name for the directory in the New name: text box.
5. Press Enter↵, or select the OK button. The directory will be renamed and the Directory Tree display will be updated.

Expanding and Collapsing Subdirectories

In Figure 2-1, the Directory Tree area is displaying the top section of the tree, so the first directory shown is the root directory, C:\. Beneath this there are six subdirectories visible: BACKUP, DIALBAN, DOS, DV, GAMES, and GV. Scrolling down will reveal more subdirectories.

When you start the Shell, only the first level of subdirectories is displayed. Subdirectories of the subdirectories are not shown. In Figure 2-1, three of the directories have a plus sign in the square brackets next to them: DOS, GAMES, and GV. The plus sign indicates that they contain subdirectories. There are several ways to expand the directory tree to show more branches. Clicking the mouse pointer on the plus sign next to a directory expands it one level to display its subdirectories, but the lower level directories will not be displayed. To expand the directory tree with the keyboard, select the directory and press + (plus); or choose the Expand One Level option on the Tree menu. (Pressing alt-T will take you to the Tree menu).

To expand all branches beneath a particular directory, select the directory name and press * (asterisk), or select the Expand Branch option on the Tree menu. To expand all of the branches in all of the directories, press Ctrl-*, or choose Expand All from the Tree menu.

When a directory is expanded, the [+] is replaced by a [–], to indicate that its subdirectories are displayed. If a directory has neither a plus nor a minus in the brackets, then it has no subdirectories.

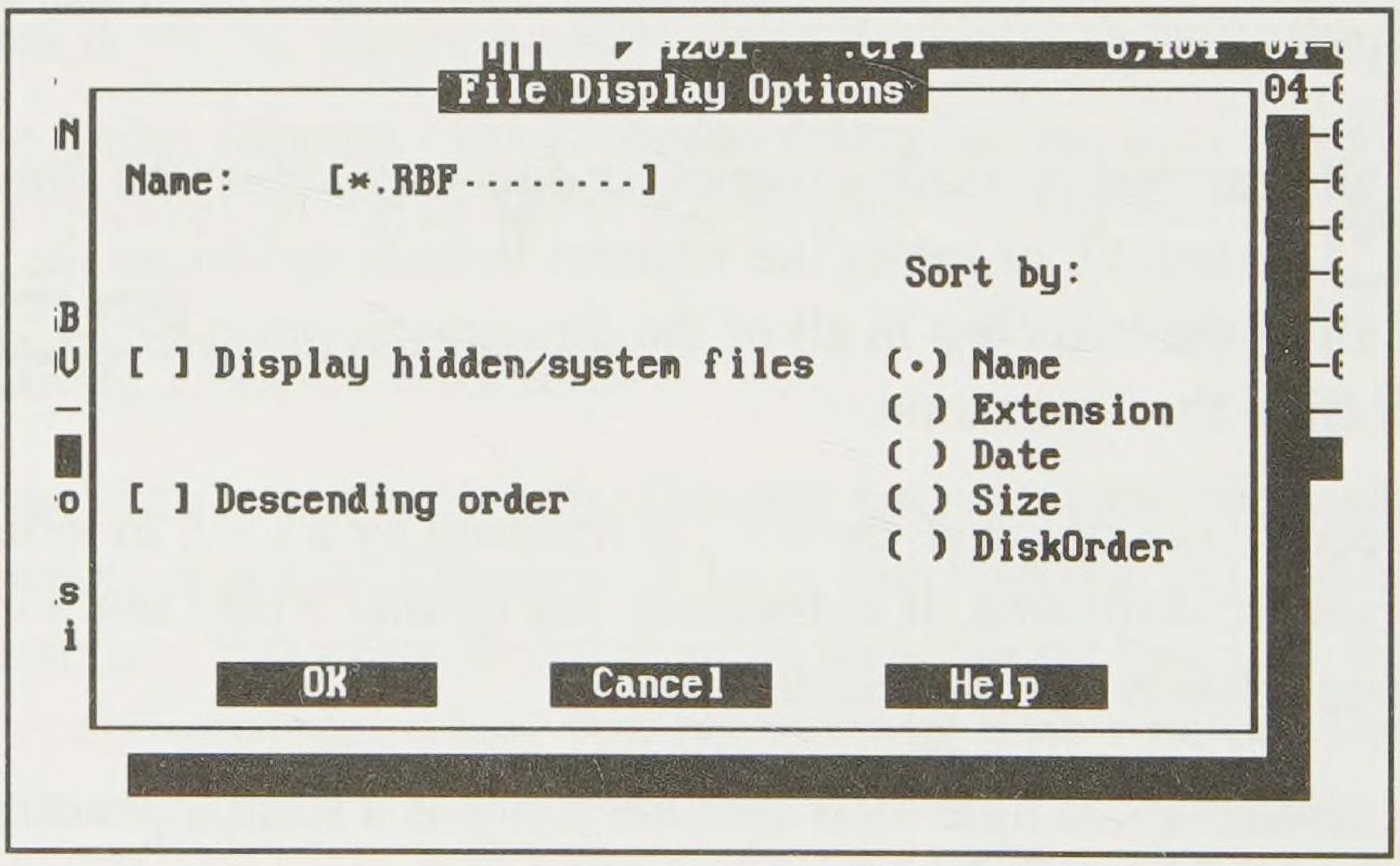
Collapsing directories to hide their subdirectories is a similar process. Selecting a directory and pressing - (minus), or clicking the mouse pointer on the [–] next to an expanded directory, causes it to collapse all of it subdirectories. Alternatively, you can choose the Collapse Branch option from the Tree menu. The [–] is replaced by a [+] to indicate there are subdirectories that are not displayed. There are no options to specify that you want to collapse a single branch versus all branches because you can highlight the level to collapse to. If you had all branches of all subdirectories expanded, and wanted to display just the first level subdirectories: select the root directory, press -, then press +. - collapses all directories beneath the root, and + expands all first-level subdirectories.

Managing Files in the Shell

When you select a directory in the Directory Tree area, the files contained in that directory are displayed in the File List area. When the Shell first appears, the File List area displays all of the files in the root directory, sorted by name. However, it is possible to change the File List display to more closely meet your own particular needs. For example, you may only want to see files with the extension of .LET, and you may want them to appear in date order.

The File List display is controlled by the File Display Options Dialog box. This box is activated by selecting the File Display Options selection on the Options menu. To access this dialog box:

1. Press alt-O to pull down the Options menu.
2. Press F to display the File Display Options dialog box (see Figure 2-12).



Name: [*.RBF........]

Sort by:

[ ] Display hidden/system files

(•) Name
( ) Extension
( ) Date
( ) Size
( ) DiskOrder

[ ] Descending order

Figure 2-12 — File Display Options Dialog Box.

The Name: text box is used to select files by name. The default value is *.* (specifies all files).

When used as parts of file name specifications, the * (asterisk) and ? (question mark) are referred to as wild cards. The * is referred to as the "many" wild card and it means: any set of characters can go here. For example, to indicate all files that have an extension of LET you would enter *.LET. This means: select files that have any name, as long as the extension is LET. Files named R.LET, R10.LET, and ROBERT.LET would all be selected. You could also select all files that begin with the letter R and have any extension by specifying R*.*, or files that begin with R and have an extension of LET by specifying R*.LET. You cannot, however, specify *R.* because anything following the * will be ignored, so *R.* selects all files.

The ? is referred to as the "single" wild card and it means: any single character can go here. For example, specifying R?.LET would select R1.LET and RA.LET, but not ROB.LET. Unlike the *, the ? can be both preceded and followed by other letters. Question mark and asterisk wild cards can be mixed in a single file specification.

Figure 2-13 shows several legal file specifiers, and some sample file names that would be selected by them.

File Specifiers	Sample Files Selected		
?OB.LET	BOB.LET	DOB.LET	FOB.LET
*.DBF	A.DBF	BOOK.DBF	XYZ.DBF
R?B.*	RAB.	RIB.X	ROB.LET
B??K*.*	BEAK.FY	BOOK.LET	BOOK12.DBF
B*.?ET	BAR.BET	B.GET	BOOK.LET

Figure 2-13 — Examples of Wild Cards in File Specifiers.

The backslash character (\ ) is used to separate the names of directories and files. Since the root directory is always named by a single backslash, you can specify the full path from the root directory to any file by starting with the drive specifier and a backslash, then naming all of the subdirectories that lead to the file — separated by backslashes — and finally specifying the file itself. For example, this is the full path name for a file named BOBSMITH.LET in the LETTERS subdirectory of the WP subdirectory of the root directory of the C: drive:

```
C:\WP\LETTERS\BOBSMITH.LET
```

On the shell screen, the title for the File List area always shows the full path and file specifier used to select files. In Figure 2-9 this is C:\ RBFILES \ BOOKS *.RBF. Thus, the current drive is C:, the directory is the root directory, and the file specifier is *.* (all files).

Beneath the Name: text box in Figure 2-12 are two check boxes. The first is labeled: Display hidden/system files. Hidden files are files whose hidden attribute is turned on. Files are usually marked as hidden because a software developer wants to protect them from tampering. They are often files that a program depends on, but that never require any interaction with a user. They are therefore marked as hidden in order to prevent curious users from moving, deleting, renaming, or otherwise changing them. System files are special hidden files that are part of the operating system. These files are part of the internal operations of DOS 5.0 and should never be modified or tampered with. There are two system files in DOS 5.0: IO.SYS and MSDOS.SYS. Hidden and system files are not displayed in the File List area unless the Display hidden/system files box is checked.

The second check box is labeled: Descending order, and is used to reverse the order in which files appear in the File List area. When the Shell first comes up, files are shown in ascending order: files whose names begin with A appear before files whose names begin with B, and so on. Checking the Descending order box will reverse this.

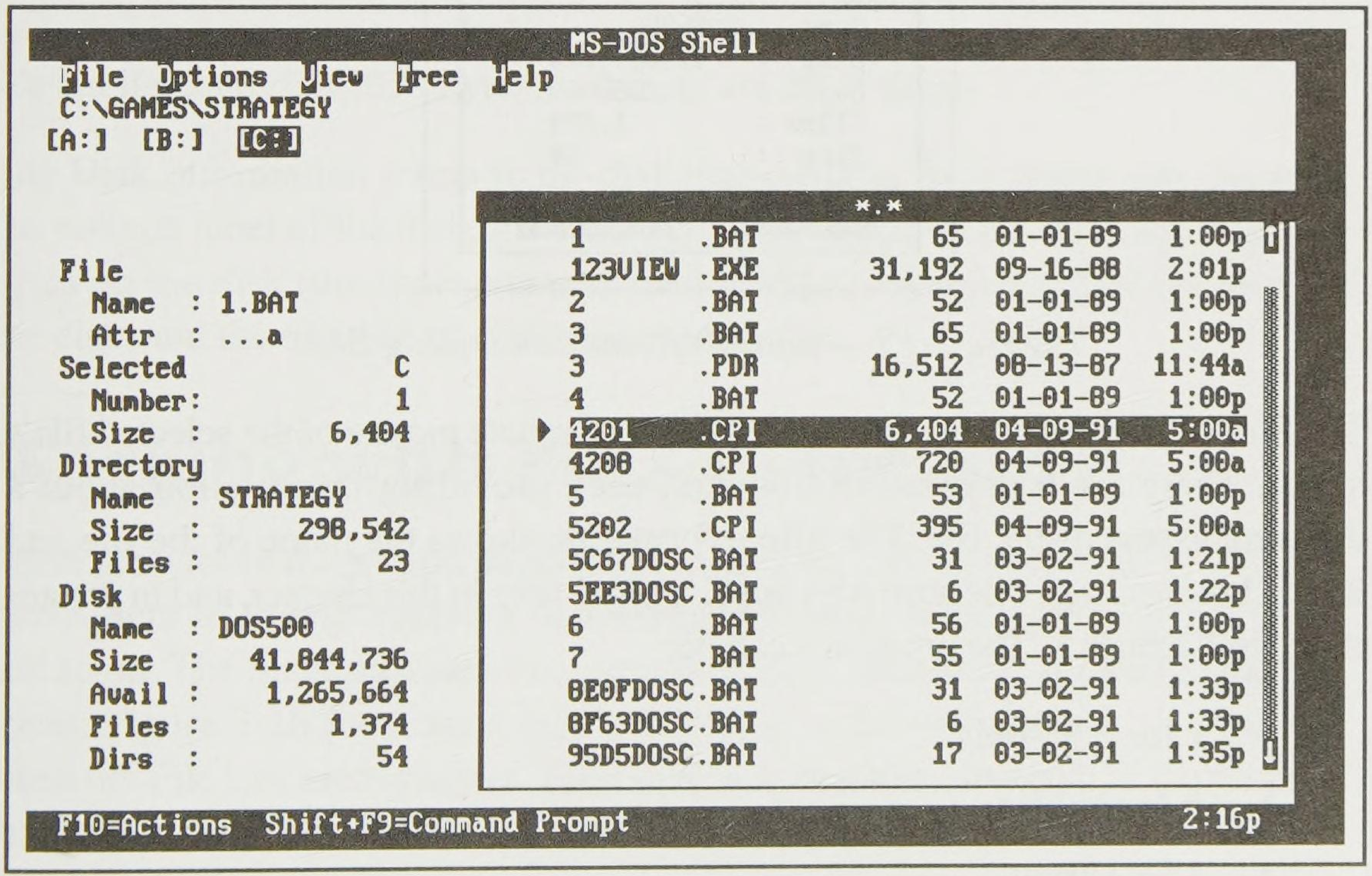
Along the right side of the File Display Options dialog box is a list of option buttons used to specify the criteria on which file names are sorted. Only one of these may be set at a time. The options and their effects are shown in Table 2-1.

Table 2-1 — Sort Options for File List Area.

Sort Option	Effect
Name	Sorts alphabetically by file name
Extension	Sorts alphabetically by file extension first, then by file name.
Date	Sorts by date, earliest first.
Size	Sorts by file size, smallest first.
DiskOrder	No sorting. Files appear in the order in which they are stored in the directory.

When all of the File List Display options have been set, select the OK button to remove the dialog box and display the new File List.

The Shell also provides a way of displaying all the files on a disk, ignoring the Directory Tree structure. This is done by choosing the All Files option from the View menu. The File Display Options dialog box is still available, so you may still manipulate the File List to include only the files you want, and sort them according to whatever criteria you want. But the files that appear now can be anywhere on the disk, as long as they meet the conditions specified in the File Display Options dialog box. When the All Files option is selected the Directory Tree is no longer relevant, so it is not displayed. This leaves some room on the screen, so the Shell adds the time of creation or last modification for each file, and displays technical information about the currently selected file. This display is shown in Figure 2-14.



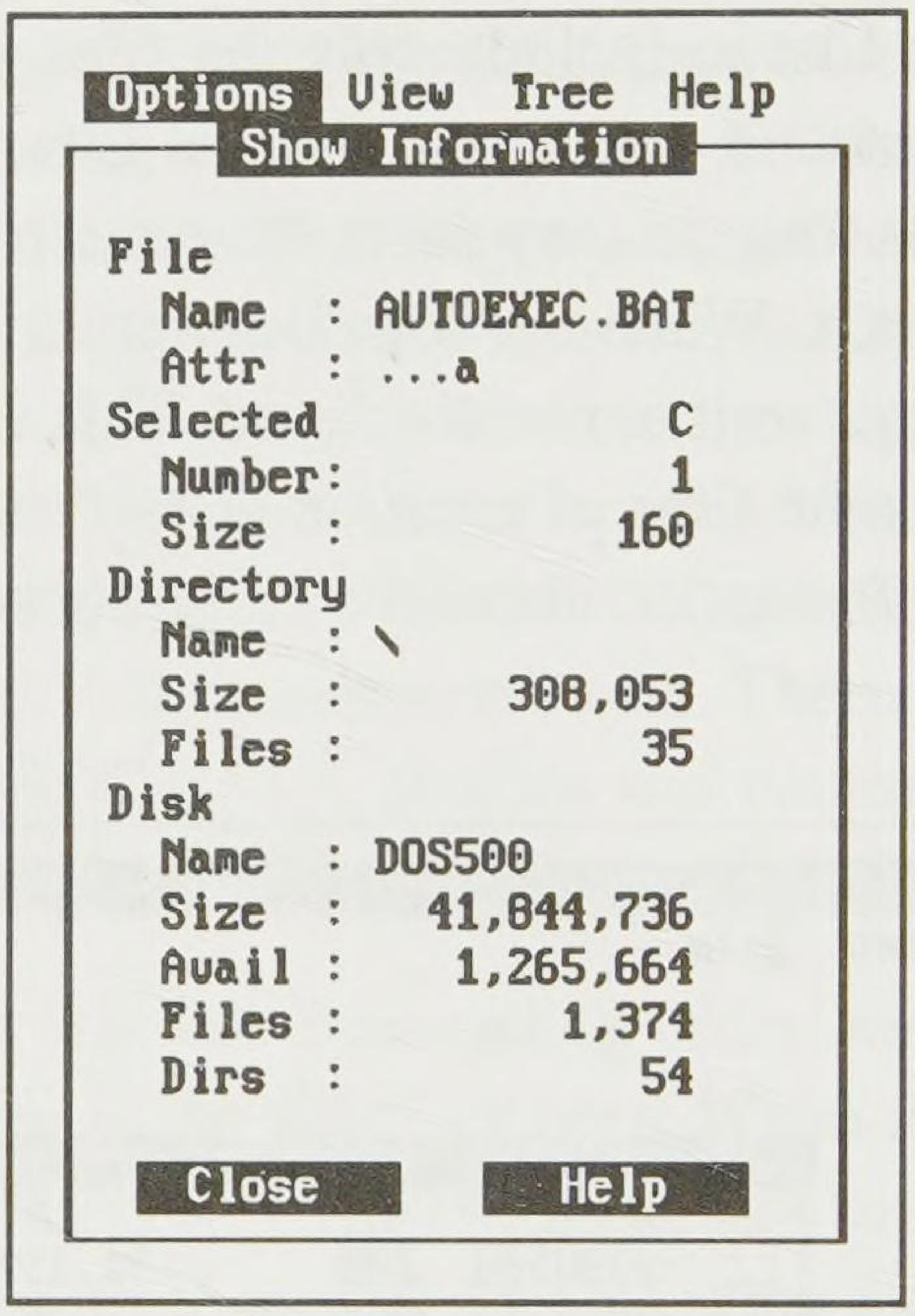
```
                                MS-DOS Shell
 File  Options  View  Tree  Help
 C:\GAMES\STRATEGY
[A:]  [B:]  [C:]

                                                   *.*
                             1       .BAT           65  01-01-89   1:00p
 File                        123VIEW .EXE       31,192  09-16-88   2:01p
   Name  : 1.BAT             2       .BAT           52  01-01-89   1:00p
   Attr  : ...a              3       .BAT           65  01-01-89   1:00p
 Selected          C         3       .PDR       16,512  08-13-87  11:44a
   Number:         1         4       .BAT           52  01-01-89   1:00p
   Size  :     6,404       ▶ 4201    .CPI        6,404  04-09-91   5:00a
 Directory                   4208    .CPI          720  04-09-91   5:00a
   Name  : STRATEGY          5       .BAT           53  01-01-89   1:00p
   Size  :   298,542         5202    .CPI          395  04-09-91   5:00a
   Files :        23         5C67DOSC.BAT           31  03-02-91   1:21p
 Disk                        5EE3DOSC.BAT            6  03-02-91   1:22p
   Name  : DOS500            6       .BAT           56  01-01-89   1:00p
   Size  :  41,844,736       7       .BAT           55  01-01-89   1:00p
   Avail :   1,265,664       8E0FDOSC.BAT           31  03-02-91   1:33p
   Files :       1,374       8F63DOSC.BAT            6  03-02-91   1:33p
   Dirs  :          54       95D5DOSC.BAT           17  03-02-91   1:35p

 F10=Actions  Shift+F9=Command Prompt                               2:16p
```

Figure 2-14 —All Files Screen.

The information on the left side of the screen is the same as that in the Show Information Dialog box, discussed in the following section.

Displaying Technical Information About a File

Detailed technical information about the currently selected file may be displayed by choosing the Show Information option on the Options menu. This displays the dialog box shown in Figure 2-15.



```
File
  Name  : AUTOEXEC.BAT
  Attr  : ...a
Selected           C
  Number:          1
  Size  :        160
Directory
  Name  : \
  Size  :    308,053
  Files :         35
Disk
  Name  : DOS500
  Size  : 41,844,736
  Avail :  1,265,664
  Files :      1,374
  Dirs  :         54
```

Figure 2-15 — Show Information Dialog Box.

The Show Information dialog box provides a complete picture of the selected file's status. There are four areas in this box, each providing information about a different aspect of the file. The **File** information shows the name of the file and its attribute settings. File attributes are discussed later in this chapter, and in greater detail in Chapter 5. The attributes can be:

- **r** for read-only
- **h** for hidden
- **s** for system
- **a** for archive

Any combination of these attributes may be set for a file. In Figure 2-15 the file name is AUTOEXEC.BAT and the only attribute that is set is (a). If more than one file is selected, the first file that was selected in the group is the one referred to here.

The **Selected** information refers collectively to all selected files. It shows the total number of selected files, and the sum of their sizes. In Figure 2-15 only one file has been selected, and its size is 160 bytes. If you have worked with two disks (displaying a Dual File List), then there will be two drive letters displayed, and beneath each one the number of selected files on that disk. The Size will show the combined sizes of selected files on both disks.

The **Directory** information indicates the directory that contains the selected file. The total number of bytes occupied by the files in the directory and the total number of files in the directory are displayed. Files in subdirectories are not included in these numbers. In Figure 2-15, the directory is the root directory, the files in it occupy 308,053 bytes, and there are 35 of them.

The **Disk** information refers to the disk that contains the selected file. It shows the volume label of the disk, the total number of bytes on the disk, the available bytes on the disk (the space not occupied by files), the total number of files on the disk, and the number of directories on the disk.

Working with Files in the Shell

There are a lot of things you might want to do with files on your disk: copy them, move them from one directory to another, run them (if they are program files), and so on. The Shell provides easy access to all of these actions through the File menu. Figure 2-16 provides a quick reference to the options that are available when the File List area is active. Each option is described in detail in the sections that follow.

File Menu Option	**Effect**
Open	If an executable file is selected, Open runs it. If a non-executable file that is associated with an executable file is selected, Open runs the associated executable file with the selected file passed as a parameter.
Run	Provides a dialog box for entering a DOS command or running a program.
Print	Adds a file to the print queue.
Associate	Associates a file extension with an executable file.
Search	Determines the location of specified files on the disk.
View File Contents	Displays the contents of a file. Permits switching between ASCII and hex displays.
Move	Moves selected files from one disk and/or directory to another.
Copy	Copies selected files from one disk and/or directory to another.
Delete	Deletes selected files from the disk.
Rename	Renames selected files.
Change Attributes	Changes the attributes of selected files.
Select All	Selects all files displayed in the File List area.
Deselect All	Deselects all files displayed in the File List area.
Exit	Exits the Shell permanently.

Figure 2-16 — File Menu Options for the File List Area.

Selecting Files

A file must be selected (or highlighted) before the Shell can interact with it. You may select an individual file or a set of files. Selecting a set of files allows you to manipulate multiple files in a single step. There are several ways of selecting files, depending on whether you are using a mouse; you are selecting a single file or multiple files; and, if you are selecting multiple files, whether those files appear in sequence in the File List or are scattered throughout it.

You select a single file with a mouse by moving the mouse pointer to the file name and clicking once. You may need to use the scroll bar to bring the desired file into view. With the keyboard, a file may be selected either by using the arrow keys to move the selection cursor to the desired file, or by pressing the first letter of the file name, which causes the selection cursor to jump to the first file that begins with that letter. If there is more than one file that begins with the letter, pressing the letter multiple times moves the selection cursor through them sequentially. You may jump immediately to the end of the File List by pressing end, or to the beginning by pressing home.

To select a group of sequential files with a mouse:

1. Click on the first file in the group.
2. Hold down ⇧ Shift and click on the last file in the group. All of the files between the first and last one, inclusive, will be selected.
3. You may adjust the group by holding down ⇧ Shift and clicking on another file name. The group of selected files will change to contain the files between the first one selected and the last file you clicked on.

With the keyboard, you can select a group by moving the selection cursor to the first file in the group, then holding down ⇧ Shift and using the arrow keys to move the selection cursor. The end of the selected group will move with the selection cursor, selecting all files between the cursor and the first file selected.

Sets of files that are not in sequence can be selected using Ctrl in combination with the mouse. When Ctrl is held down, you may use the mouse to click on randomly positioned files throughout the File List. Each file clicked on will be highlighted, to show that it has been added to the set of selected files. The mouse button acts as a toggle, allowing selected files to be deselected by clicking on them.

To use the keyboard for selecting files that are not in sequence involves the following steps:

1. Move the selection cursor to the first file in the set.
2. Press ⇧Shift-F8. This selects the first file and turns on the ADD mode (the word ADD will appear near the right end of the Status Bar at the bottom of the screen).
3. Move the cursor to the next file to be added to the list. You may use any of the usual ways of moving the cursor: pressing the first letter of the file you want to go to, the arrow keys, or home or end.
4. When the cursor is on the next file, press space. The file will be added to the list of selected files. space acts as a toggle, so you may deselect files by pressing it when the cursor is on a selected file.
5. Repeat steps three and four for all of the files you want to include.
6. Press the ⇧Shift-F8 to turn the ADD mode off (the word ADD will disappear from the status bar).

This technique for selecting sequential and non-sequential sets of files may be combined to select several groups of files, where the files within each group are in sequence, but the groups themselves are scattered throughout the file list. With a mouse this is accomplished as follows:

1. Select the first group of sequential files as previously described.
2. Hold down Ctrl and select the first file in the next group.
3. Now hold down Ctrl-⇧Shift and click on the last file in this group.
4. Repeat step 3 for any other groups.
5. Add individual files by holding down Ctrl and clicking on them.
6. Deselect individual files by holding down Ctrl, then clicking on them.

To accomplish this with the keyboard, carry out these steps:

1. Select the first group of sequential files as previously described.
2. Press ⇧ Shift-F8 to turn on the ADD mode.
3. Move the selection cursor to the first file in the next group with any of the usual keyboard techniques.
4. Press space to select the file.
5. Hold down ⇧ Shift and use the usual keys to move the selection cursor to the last file in the group.
6. Repeat steps 4 and 5 for any other groups.
7. Press ⇧ Shift-F8 to turn off the ADD mode.
8. You may add individual files to the set by turning on the ADD mode, moving the selection cursor to them, and pressing space.
9. You may also remove selected files from the set the same way.

Note: Whenever a group of files is selected, if you use the keyboard to move the selection cursor without having the ADD mode turned on, or use the mouse to move the selection cursor without having Ctrl pressed, you will deselect all files that have been selected. The file at the new cursor position will be the only selected file. Consequently, whatever you want to do with a set of selected files must be done before moving the selection cursor within the File List area.

STOP

Instead of manually choosing specific files to be selected, the Select All option from the File menu may be used to select all of the files displayed in the File List area. The Deselect All option will deselect any files that have been selected. There are keyboard shortcuts for these two options: pressing Ctrl-/ combination selects all files, and Ctrl-\ deselects all files. If you have a long list of sequential files with only a few that you don't want, the quickest way to select them is to select the entire list, then deselect the ones you don't want.

SHORTCUT

In summary, using Ctrl with the mouse, and using the ADD mode with the keyboard, both mean: Save whatever's been selected and continue selecting more, while ⇧ Shift says: Select everything between the last selected file and the next selected file. Ctrl-/ selects all files, and Ctrl-\ deselects all files.

SHORTCUT

Another technique to streamline the file selection process is to first use the File Display Options choice, on the Options menu, to display a subset of the files in the current directory. The selection process will then be limited to those files.

The Options menu has another choice that is related to File Selection: Select Across Directories. This option is a switch that is either on or off. When the Shell starts up it is off — you must turn it on by selecting it. When Select Across Directories is on, a small diamond will be shown next to it on the menu.

LoOk

When Select Across Directories is off, changing directories automatically deselects any files that have been selected. For example, if you go into a directory named WP and select all the files, then change the current directory to the root directory, all of the files in WP will be automatically deselected. Turning Select Across Directories on causes selected files to remain selected, regardless of which directory is currently displayed. This is quite risky because it is very easy to leave files marked in a directory and not realize it. If you delete or move all selected files, then those files will be unintentionally deleted or moved.

You may also use the All Files display, selected on the View menu, to work with all files on the disk. When the All Files view is displayed, the File List area shows all files on the disk with no regard for the directory tree structure. Consequently, this view always behaves as though Select Across Directories is on.

Associating Data Files with Program Files

Many programs accept the name of a data file as a parameter when you call the program from the command line. This causes the data file to be loaded automatically when the program starts. This is very common with word processors, spreadsheets, and database management systems. For example, if the WP command will start your word processor, then entering

```
WP BOBSMITH.LET
```

at the command prompt will bring up the word processor with the file BOBSMITH.LET loaded and ready for editing. This saves several steps over starting the word processor, then loading the file.

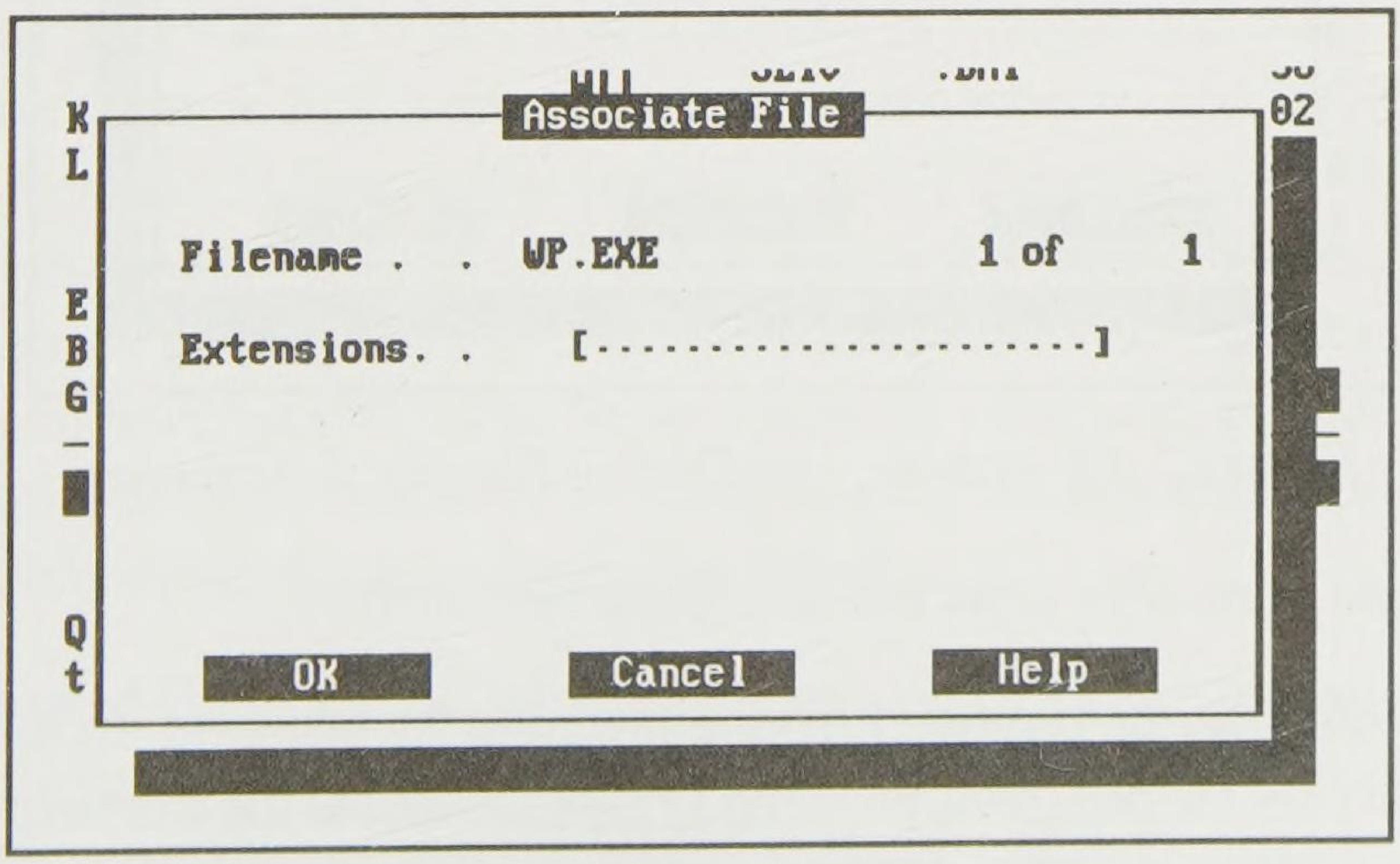
You can streamline this process even more within the Shell, by using the Associate feature. If you name the files that contain your correspondence with the extension LET, you can associate that extension with your word processing program. Opening any file that has the LET extension then runs the word processor, passing the selected file as a parameter. There are three ways to open a file:

1. Click on it with the mouse.
2. Highlight it with the selection cursor and press Enter←.
3. Highlight it with the selection cursor and select Open from the File menu.

There are two main advantages gained by associating data files with their program files: it is faster than running a program and then opening the data file; and it allows you to run the program from whatever directory a data file is in, with a single keystroke or mouse click.

A program may have more than one extension associated with it. For example, if the extension LET is used for correspondence files, and the extension RPT for reports written with a word processor, then both of these extensions could be associated with the word processor. On the other hand, any given extension can be associated only with a single program. The LET extension could not be associated with a word processor and a database program.

To associate extensions with a program, select the program in the File List and choose Associate from the File menu. The dialog box shown in Figure 2-17 will be displayed.



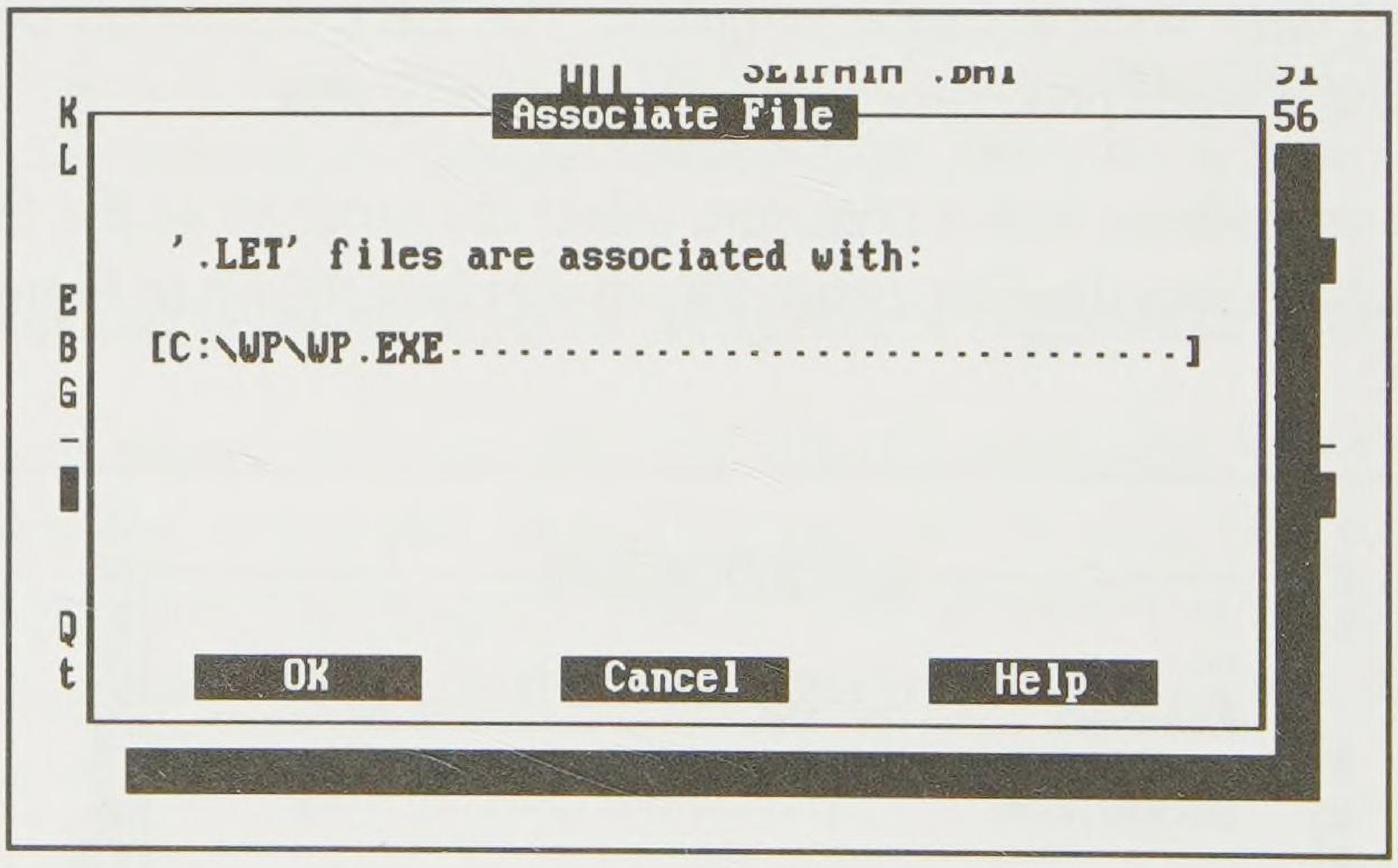
```
Filename . .  WP.EXE                1 of     1
Extensions. .    [......................]
```

Figure 2-17 — The Associate File Dialog Box.

The extensions to be associated with the program are entered into the text box labeled Extensions. Selecting OK, or pressing Enter←, completes the process. Selecting Cancel, or pressing esc, aborts it.

If there are already extensions associated with the program, they will appear in the Extensions box. If you want to replace them, simply start typing the new ones. The existing ones disappear and the new ones are entered on the blank line. If you want to edit the existing extensions, press → or ← before any other key. ← will move the cursor to the beginning of the text box, but will leave the existing extension(s) intact. → will leave the cursor in position at the end of the existing extensions, but now anything you type will be added to the list.

You may also use the Associate option to relate (or associate) a program to a specific file extension. This is done by moving the selection cursor in the File List area to a file with the extension of interest, and choosing the Associate option on the File menu. If the LET extension is associated with WP.EXE, then selecting a file with a LET extension and choosing Associate will bring up the dialog box shown in Figure 2-18.



Associate File

'.LET' files are associated with:

[C:\WP\WP.EXE.............................]

Figure 2-18—Associate Dialog Box for Extensions.

You can create or modify associations, using this method.

All associations may be removed from a program by bringing up the Associate File dialog box for the program, pressing ←BkSp to delete the existing extensions, then selecting OK or pressing Enter←. Alternatively, a particular extension may be removed by bringing up the Associate File dialog box for a file with that particular extension, pressing ←BkSp to delete the association, then selecting OK or pressing Enter←.

Opening Files

The Open option on the Files menu is used to open, or start, the currently selected file in the File List. If the currently selected file is an executable file (if it has an extension of BAT, COM, or EXE) then the file itself is started. If the file is not executable, but is associated with a program file, then the associated program file is run with the selected file passed as a parameter. If the selected file is neither executable nor associated with an executable file, then Open simply causes the computer to beep, indicating that it cannot open the selected file.

Printing Files from the Shell

The Print option on the File menu is used to send all selected files to the printer. Print can be selected when one or more files are selected. If no files are selected, the Print option is shaded, indicating that it is not available.

There are only certain types of files that can be printed with the Print option. These are known as ASCII text files, DOS text files, or unformatted files (all three terms mean the same thing). The acronym ASCII stands for **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange, and refers to a special set of 128 codes used to represent text characters (like letters, numbers, and punctuation marks) in computers. If a file contains only ASCII codes it will appear as normal text when you print it. Most programs, including word processors, spreadsheets, and database systems, do not produce ASCII files unless you specifically direct them to. Otherwise they produce formatted files. A formatted file is one that contains special codes that tell the program how to display different parts of the file. For example, a word processor will have special codes for underlining, centering, boldface, and so on. Spreadsheets and databases store their information in encoded formats that must be interpreted before the data can be displayed.

These formatted files only make sense to the programs that produced them, and consequently cannot be printed properly with the Shell's Print option. ASCII files are files that contain no special codes — they consist only of the standard alphabetic symbols. EDIT and EDLIN, the two editors that come with DOS 5.0, produce unformatted files. Most programs that produce formatted files also have options to produce ASCII files.

Print queue

The Print option makes use of the DOS print queue. A print queue is an internal buffer that stores the names of files to be printed and automatically sends the files to the printer as it is ready for them. This is convenient because it means that you do not have to wait for your files to finish printing before you can use your computer again. You can continue working and let the PRINT function send files to the printer in the background.

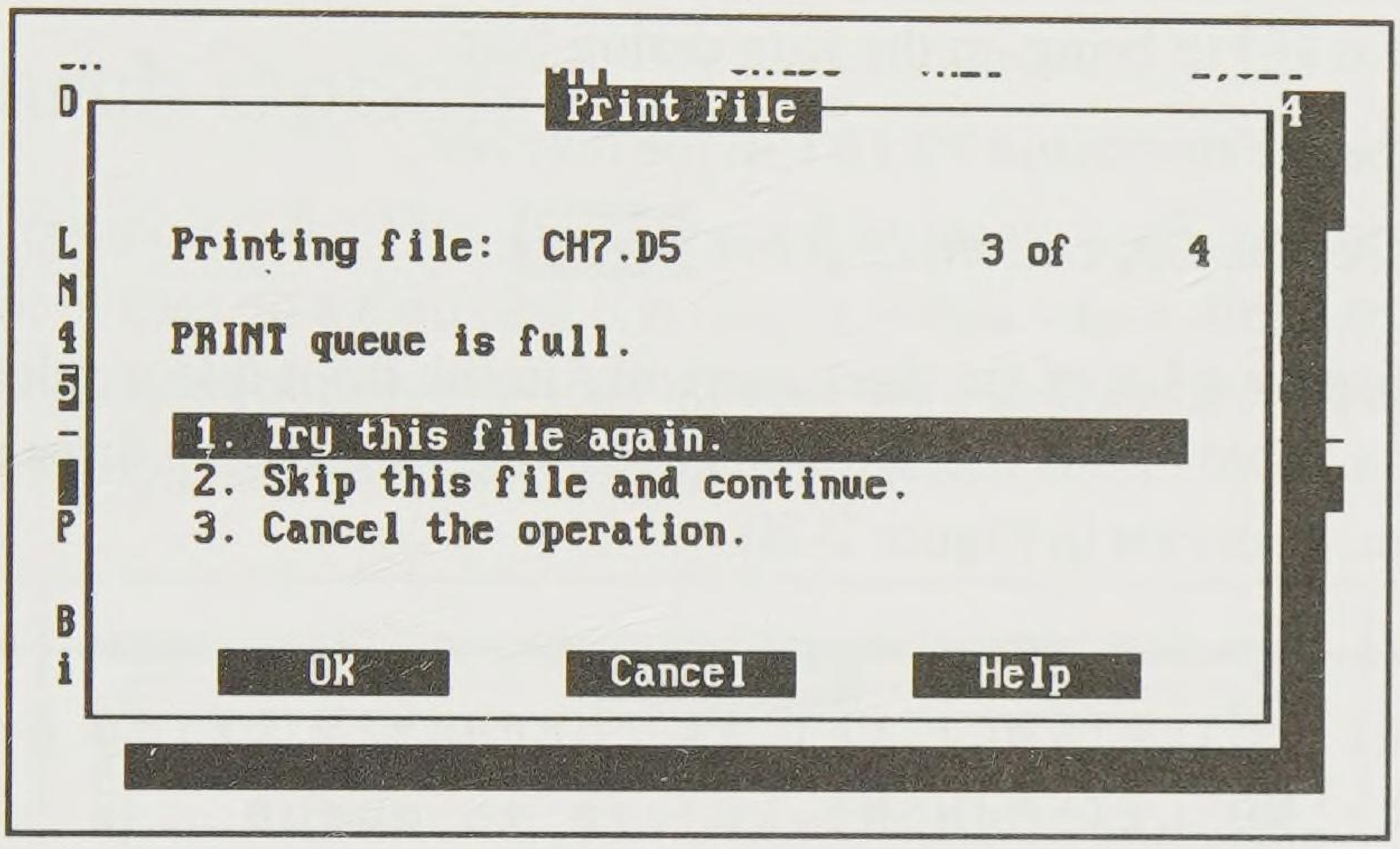
If you are working while the queue prints your files, you may notice that the computer pauses occasionally. This is because the printer has finished printing the last information sent to it, and DOS needs to take over the CPU momentarily in order to send more information to the printer.

There are several parameters that can be set to adjust the behavior and capacity of the print queue. Managing the PRINT queue is discussed in detail in Chapter 9. For now, what is important to know is that the PRINT command must be executed at the command prompt before the PRINT option can be used. Executing the PRINT command defines an area of memory to use as the print queue. If the PRINT command is not run automatically by your AUTOEXEC.BAT file, you can run it from within the Shell by selecting the RUN option on the File menu, and entering the following command in the text box:

```
PRINT /D:LPT1
```

This sets up a print queue that can hold up to ten file names, and tells the print queue that your printer is connected to a port named LPT1. (External devices that are connected to your computer, such as printers, communicate with it through connections known as "ports." Each port on a computer has a name by which it is identified.) LPT1 is the most common port used for printers. If the PRINT command does not work with this setting, refer to Chapter 9 for details on the PRINT command.

If more than 10 files are submitted to the print queue, a screen similar to that in Figure 2-19 will be displayed.



Printing file: CH7.D5 3 of 4

PRINT queue is full.

1. Try this file again.
2. Skip this file and continue.
3. Cancel the operation.

Figure 2-19 — Full Print Queue Dialog Box.

To see the dialog box in Figure 2-19, submit 10 files to the queue, to fill it up. Then submit four more files to the queue before the first file has finished printing. A similar box would appear if a single list of more than ten files had been submitted.

When this happens, you have three options:

Option 1 — You can wait for the first file in the queue to finish printing, thereby freeing up a space in the queue, and select option 1, Try this file again. The next file will then be submitted to the queue. This must be repeated for each file left until they have all been submitted. Until the last file in the list has been submitted you will not be able to use the computer for anything else.

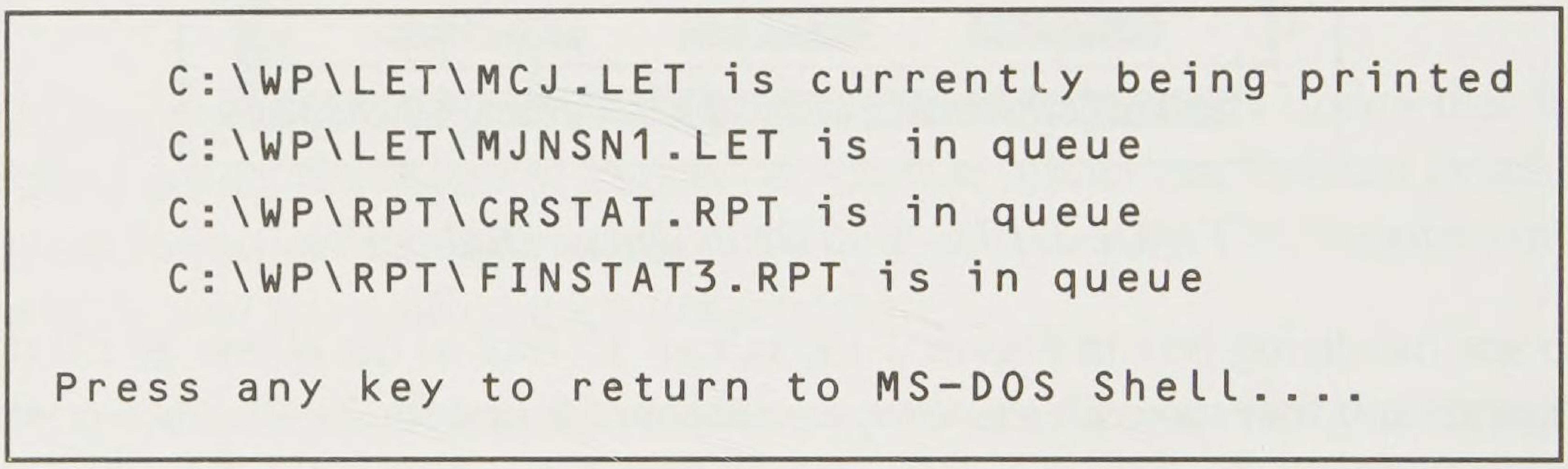
Option 2 — Skip this file and continue — removes the displayed file name from the submission list and moves on to the next one. By choosing this option for each file left in the list, you cancel the print request for each of these files. This gets you back to the Shell where you can continue working.

Option 3 — Cancel the operation — stops the attempt to submit the list of files to the queue and returns you to the Shell.

The contents of the print queue may be viewed by following these steps:

1. Press alt-F to bring up the File menu.
2. Press R to bring up the Run dialog box.
3. Type the command `PRINT` in the text box.
4. Select the OK button, or press Enter←.

This will display a list of the files currently in the print queue, along with their status. An example of this list, with one file being printed and three more waiting in the queue, is shown in Figure 2-20.



```
    C:\WP\LET\MCJ.LET is currently being printed
    C:\WP\LET\MJNSN1.LET is in queue
    C:\WP\RPT\CRSTAT.RPT is in queue
    C:\WP\RPT\FINSTAT3.RPT is in queue

Press any key to return to MS-DOS Shell....
```

Figure 2-20 — Sample Print Queue List.

Printing with copy

The print queue takes up memory that could otherwise be used by programs, even if the queue is not being used. Once the Print command is executed the first time, the queue is permanently installed until the computer is rebooted. An alternative to using the Print option is the Copy command. Copy can be used to copy a file to the printer. This requires entering the Copy command at the command prompt, or in the text box for the Run option. You cannot do this with the Copy option on the File menu. To copy a file to the printer, the following command should be entered on the command line:

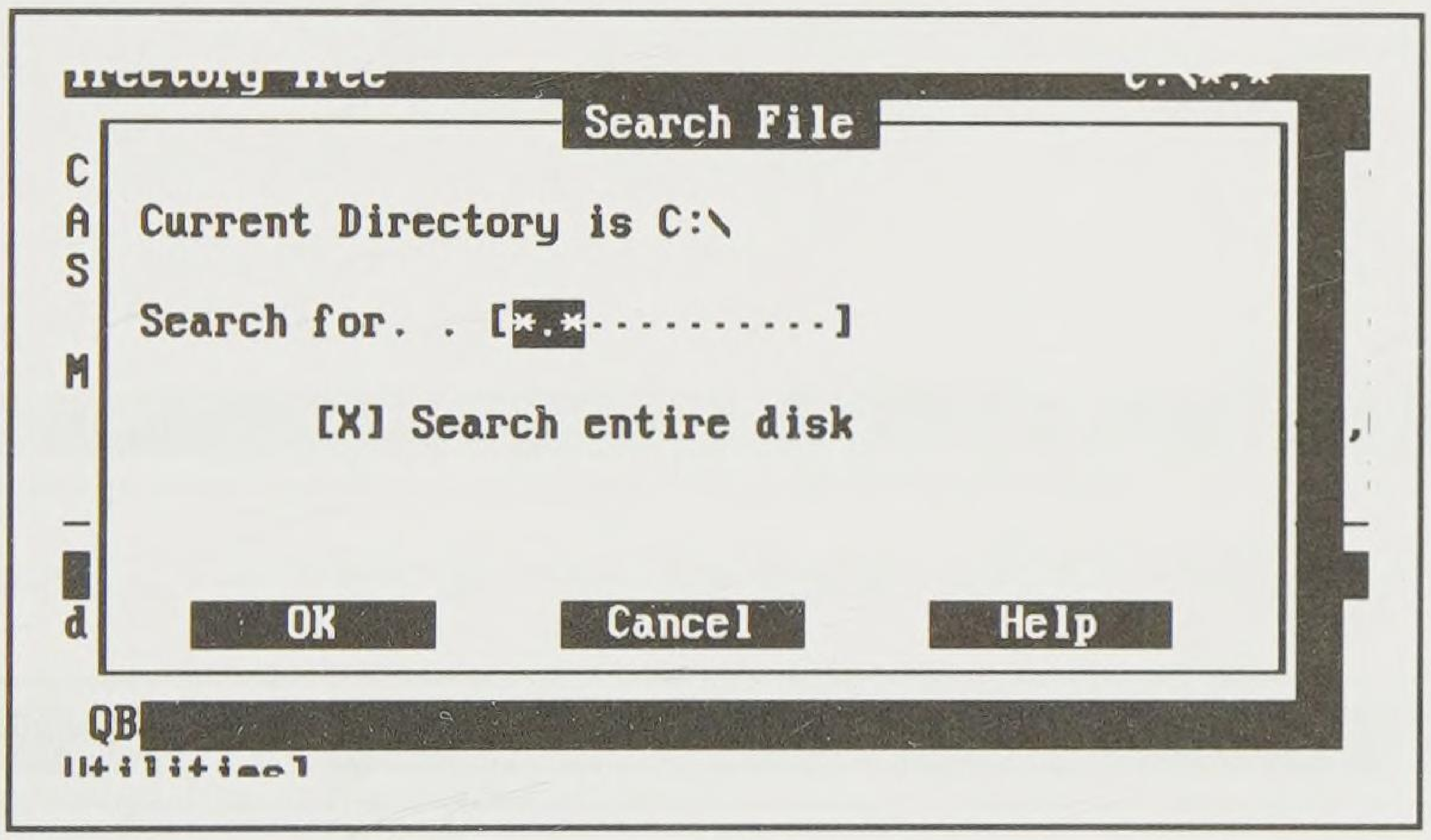
```
COPY Filename PRN
```

If the file is not in the current directory, then you must include the path specification as part of the file name or switch to that directory.

There are two disadvantages to using the COPY command instead of the print queue: you must wait for the file to finish printing before you can use the computer to do anything else, and you must execute it once for each file you want to print.

The Search Option

The Search option on the File menu is used to locate a file on the disk. When there are a lot of files on a hard disk it is easy to forget which directory a particular file is in. The Search option provides an easy way to find files. When Search is selected, the dialog box in Figure 2-21 is displayed.



Search File

Current Directory is C:\

Search for. . [*.*..........]

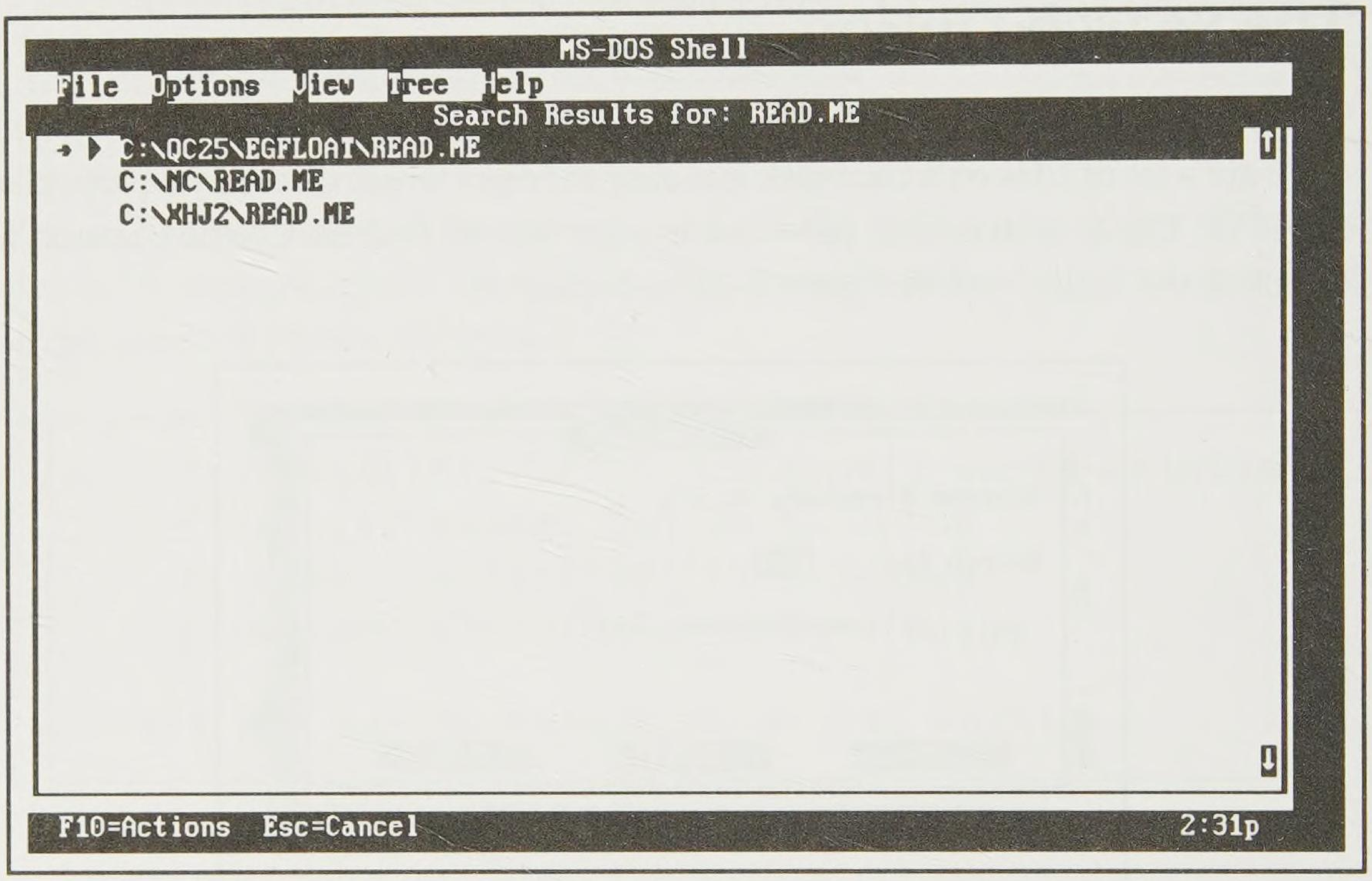
[X] Search entire disk

Figure 2-21 — Dialog Box for Search File Function.

The currently active directory is displayed at the top of the Search File dialog box. This is the directory selected in the Directory Tree area. The "Search for" text box is used to enter the name of the file or files to search for. You may enter either a single file name here, or use the file name wild cards, asterisk (*) and question mark (?), to indicate groups of files to search for.

The "Search entire disk" check box is used to specify where on the disk the Shell should look for the file(s). There are two options: if this box is not checked, only the current directory is searched. If this box is checked, then the entire disk is searched. Normally you would want this box checked, and this is the default.

After completing the search, a "Search Results" screen is displayed, similar to the one shown in Figure 2-22. This screen shows the results after the entire disk was searched for files named "read.me."



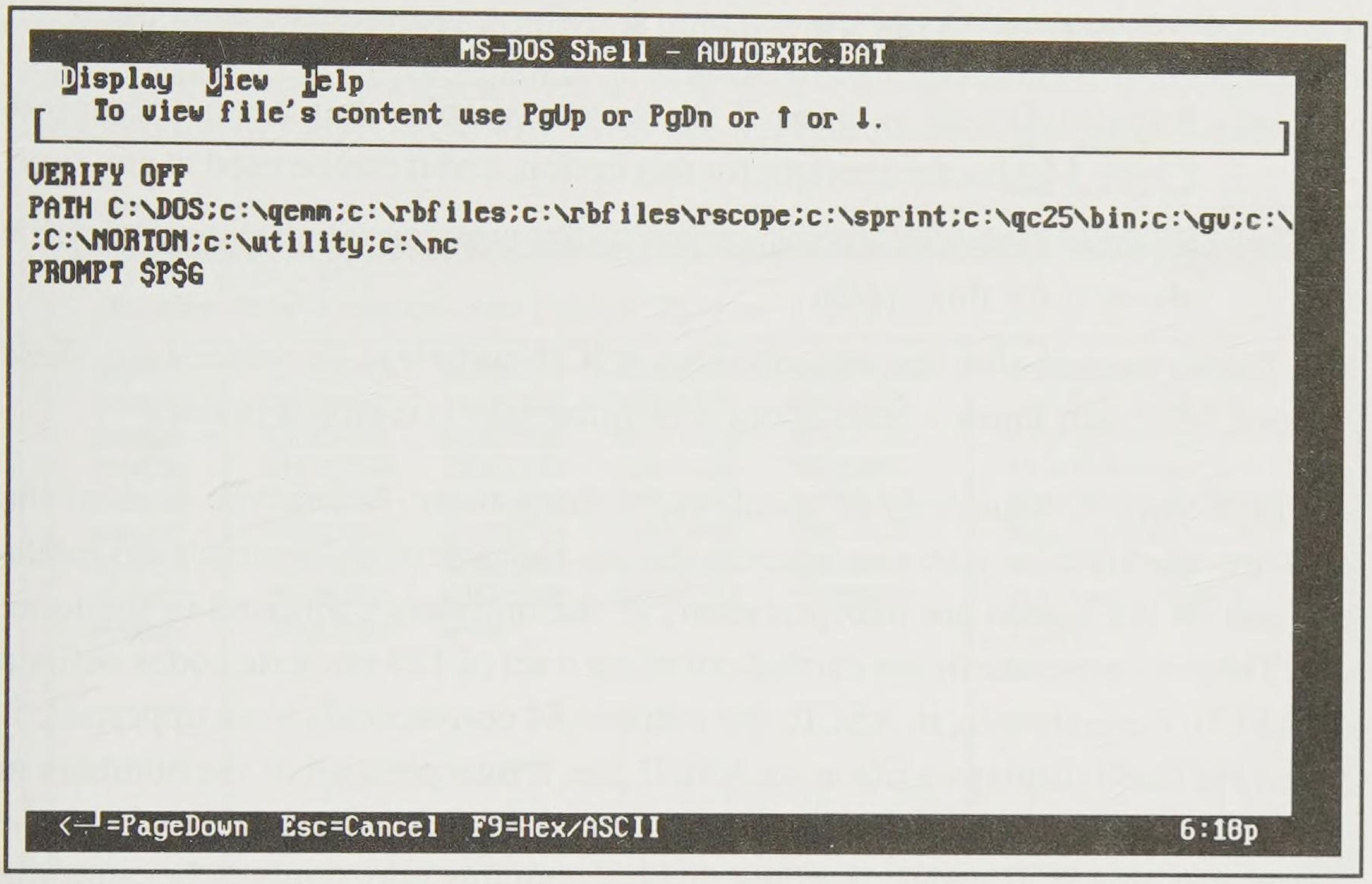
```
Search Results for: READ.ME
C:\QC25\EGFLOAT\READ.ME
C:\NC\READ.ME
C:\XHJ2\READ.ME
```

Figure 2-22 — Search Results Screen.

The View Option

The View option on the File menu is used to examine the contents of a file. It is intended to give you a quick way of checking a file's contents; it is not meant to be used for reading or reviewing files. You may want to make such a quick check before deleting a file, or to identify a file that you want to copy. You will normally only want to view text files — ones that only contain ASCII characters.

An example of an ASCII file is the AUTOEXEC.BAT file in the root directory of your hard drive. To see the contents of AUTOEXEC.BAT, select it in the File List area, then select the View option on the File List menu, or press F9 (a shortcut for the View option). A screen similar to that in Figure 2-23 will be displayed.


MS-DOS Shell - AUTOEXEC.BAT
Display View Help
To view file's content use PgUp or PgDn or ↑ or ↓.
<┘=PageDown Esc=Cancel F9=Hex/ASCII
6:10p

```
VERIFY OFF
PATH C:\DOS;c:\qemm;c:\rbfiles;c:\rbfiles\rscope;c:\sprint;c:\qc25\bin;c:\gv;c:\
;C:\NORTON;c:\utility;c:\nc
PROMPT $P$G
```

Figure 2-23 — View Screen Displaying AUTOEXEC.BAT File.

If the file contains more information than will fit on the screen, you may scroll through it in any of the following ways:

1. Press PgUp and PgDn to scroll one screen at a time.
2. Press Enter←┘ to scroll down one screen at a time.
3. Press the ↑ and ↓ keys to scroll one line at a time.
4. Click the mouse on the PgUp, PgDn, or arrow symbols at the top of the screen to achieve the same effect as pressing the keys.

There is a menu bar at the top of the screen with three options: Display, View, and Help. This menu bar is activated in the same way as the Shell menu bar; by pressing alt or F10. The Help pull-down menu provides the same options discussed earlier in this chapter, in the section titled, "Getting Online Help."

There are two options on the View menu: Repaint Screen and Restore View:

- Repaint Screen is used if the screen display has been corrupted. ⇧Shift-F5 is the shortcut for this option, and it can be used at any time.
- Restore View option returns you to the previous Shell view. esc is the shortcut for this option.

The Display menu also has two options: ASCII and Hex. To understand these options, you must know a little about how information is stored in files.

All files contain sequences of numbers, nothing more. When you look at the contents of a letter in your word processor, the characters and punctuation marks you see on the screen are interpretations of the numbers contained in the letter file. These interpretations are carried out using a set of 128 numeric codes defined by ASCII. For example, in ASCII, the number 84 corresponds to an uppercase T. When the Shell displays a file as an ASCII file, it interprets all of the numbers in the file as ASCII codes. If the file contains numbers that were not intended to be interpreted as ASCII codes, then the display contains odd symbols because the Shell cannot interpret them correctly.

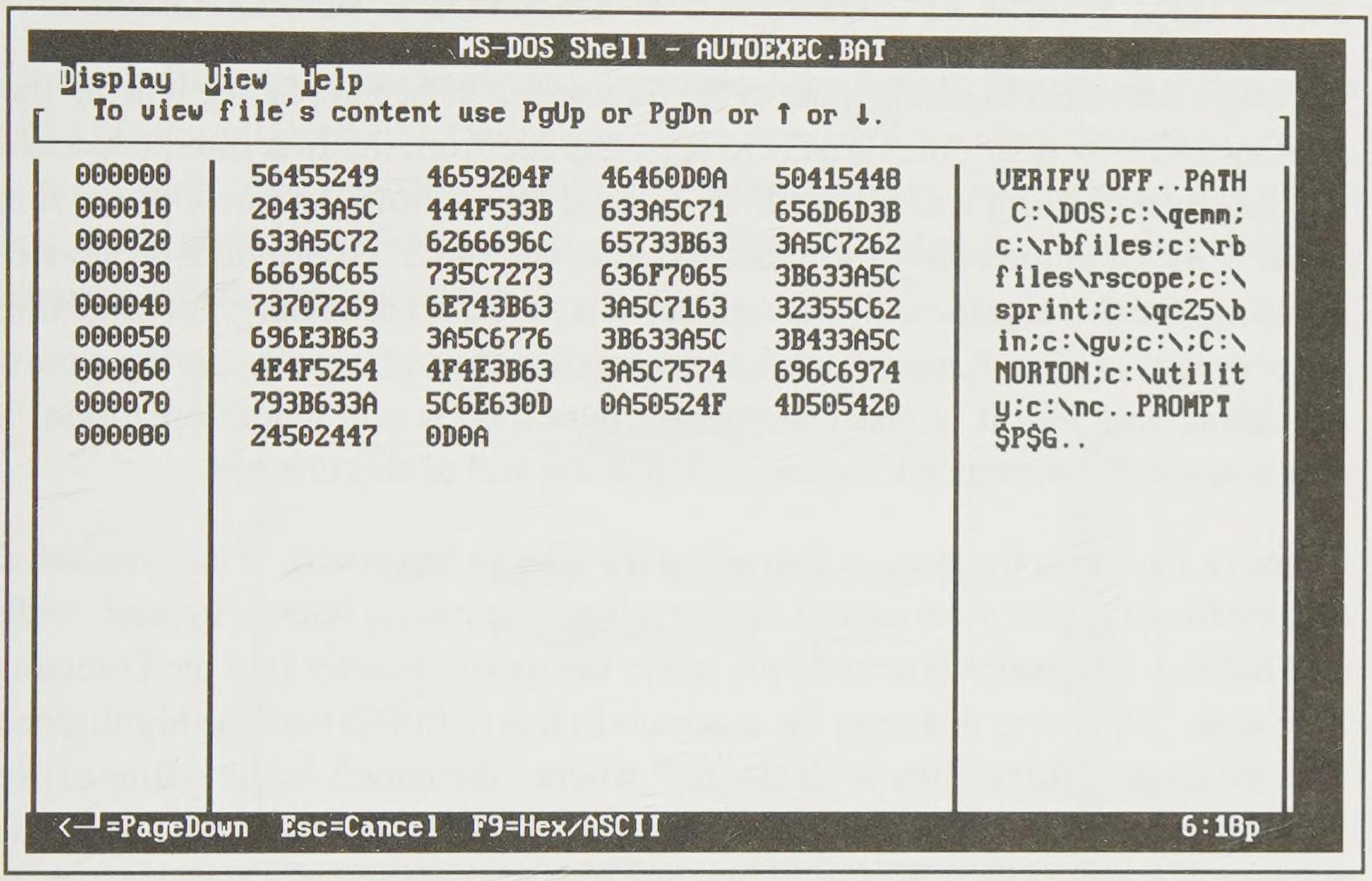
There are several reasons files may have non-ASCII codes:

- They may have been produced by a word processor that inserts special formatting codes into its files
- They may have been created by a program that encodes data before storing it
- They may not be data files at all.

Executable program files — those that have COM and EXE extensions in their names — do not use ASCII codes. Instead they use numeric codes that the computer interprets as instructions to perform specific tasks.

Whatever the reason for the non-ASCII codes, it is more useful to see their numeric representation than the symbols they produce on the screen. A knowledgeable programmer can interpret these numbers and learn something about the file being viewed. The Hex option on the Display pull-down menu displays the contents of a file as the numeric codes instead of ASCII. They are not, however, displayed in the base ten, or decimal, number system. They are displayed in the base 16, or

Hexadecimal, number system. This is closer to the way the computer actually sees these numbers, and consequently it is easier for programmers to interpret. The Hex version of the AUTOEXEC.BAT file in Figure 2-23 is shown in Figure 2-24.


MS-DOS Shell - AUTOEXEC.BAT
Display View Help
<—┘=PageDown Esc=Cancel F9=Hex/ASCII
6:18p

To view file's content use PgUp or PgDn or ↑ or ↓.

```
000000   56455249  4659204F  46460D0A  50415448   VERIFY OFF..PATH
000010   20433A5C  444F533B  633A5C71  656D6D3B    C:\DOS;c:\qemm;
000020   633A5C72  6266696C  65733B63  3A5C7262   c:\rbfiles;c:\rb
000030   66696C65  735C7273  636F7065  3B633A5C   files\rscope;c:\
000040   73707269  6E743B63  3A5C7163  32355C62   sprint;c:\qc25\b
000050   696E3B63  3A5C6776  3B633A5C  3B433A5C   in;c:\gv;c:\;C:\
000060   4E4F5254  4F4E3B63  3A5C7574  696C6974   NORTON;c:\utilit
000070   793B633A  5C6E630D  0A50524F  4D505420   y;c:\nc..PROMPT
000080   24502447  0D0A                           $P$G..
```

Figure 2-24 — Hex Version of AUTOEXEC.BAT File.

Even when the Hex mode is selected, the ASCII interpretation of the codes will still be displayed in a vertical box at the right side of the screen. This is useful because executable programs will often have strings of ASCII codes embedded in them. These strings may contain messages the program displays on the screen. This can be demonstrated by Viewing the file COMMAND.COM that comes with DOS 5.0. There are meaningful messages interspersed with meaningless symbols.

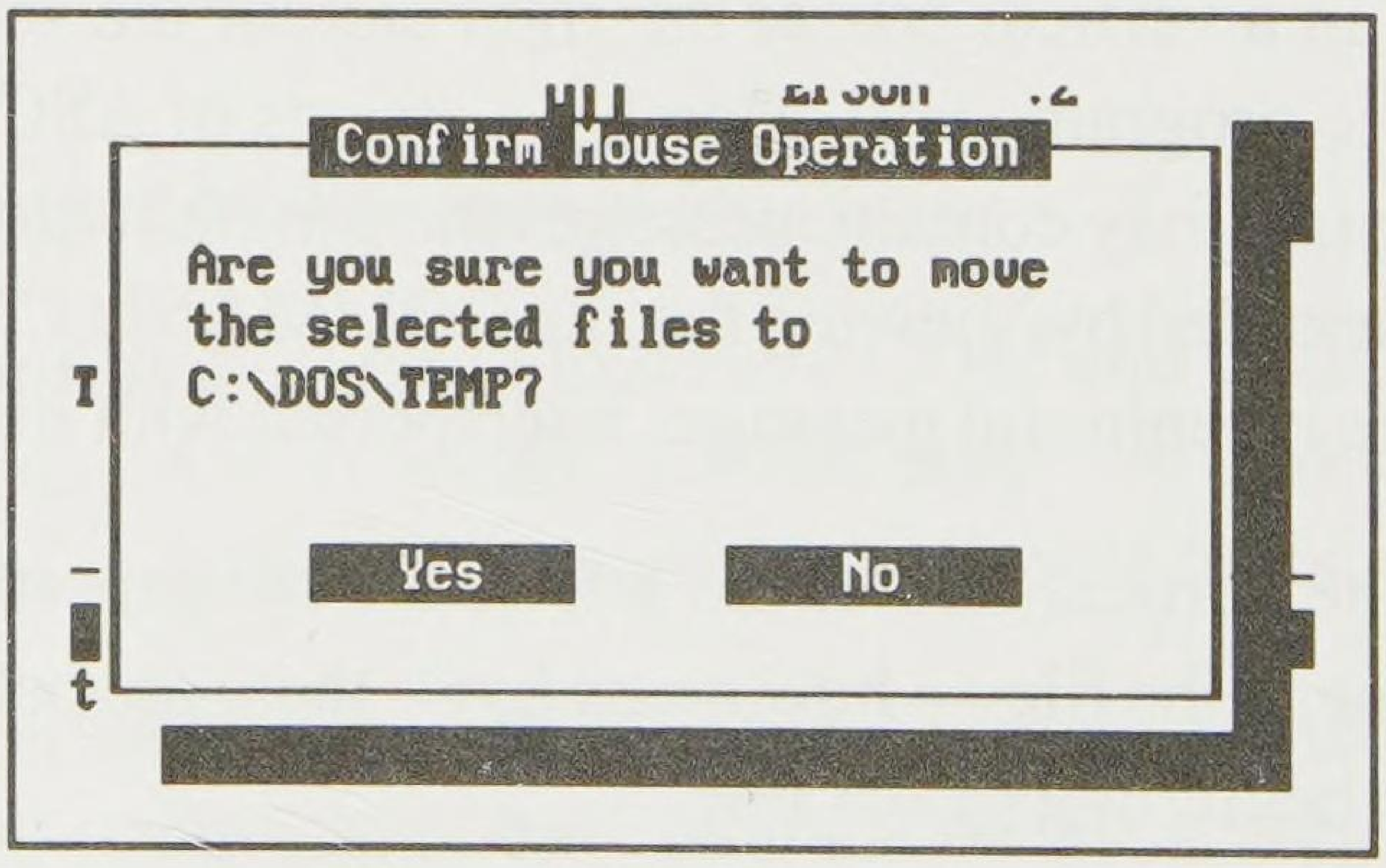
The numbers in the vertical box at the left of the screen represent the physical location of each line in the file — how many bytes there are between the beginning of the file and the beginning of the line.

When you use the View option to display the contents of a file, the Shell automatically determines the initial mode. If the Shell interprets the file as containing 100 percent ASCII symbols, then it displays the file in ASCII format. If it interprets any part of the file as non-ASCII code, it displays the file in Hex format. F9, or the Display menu, may be used to switch back and forth between these two formats.

Moving Files from One Directory to Another

Moving a file refers to taking it out of the directory it is in, and placing it in another directory. One way of doing this is to copy the file from the first directory to the second, then deleting it from the first. In fact, this is how you must move files when working at the command prompt. However, the Shell provides a one-step move feature that significantly streamlines this process, particularly for large files or for groups of files. A more technical description of what happens during a move operation, and why it is faster and better than a copy and delete operation, is explained in the Advanced Topics section at the end of this chapter.

To move files with the mouse, first select the files to be moved. When you select the last file in the set to be moved, do not release the mouse button. Instead, while still holding the mouse button down, move the mouse pointer into the Directory Tree area. Whichever directory the mouse is in line with will become highlighted. The message "Move files to dirname," where "dirname" is the name of the highlighted directory, will appear in the lower left corner of the screen. Now, still holding down the mouse button, move the mouse pointer to the target directory for the selected files and release the mouse button. A dialog box like the one shown in Figure 2-25 will be displayed.


Confirm Mouse Operation

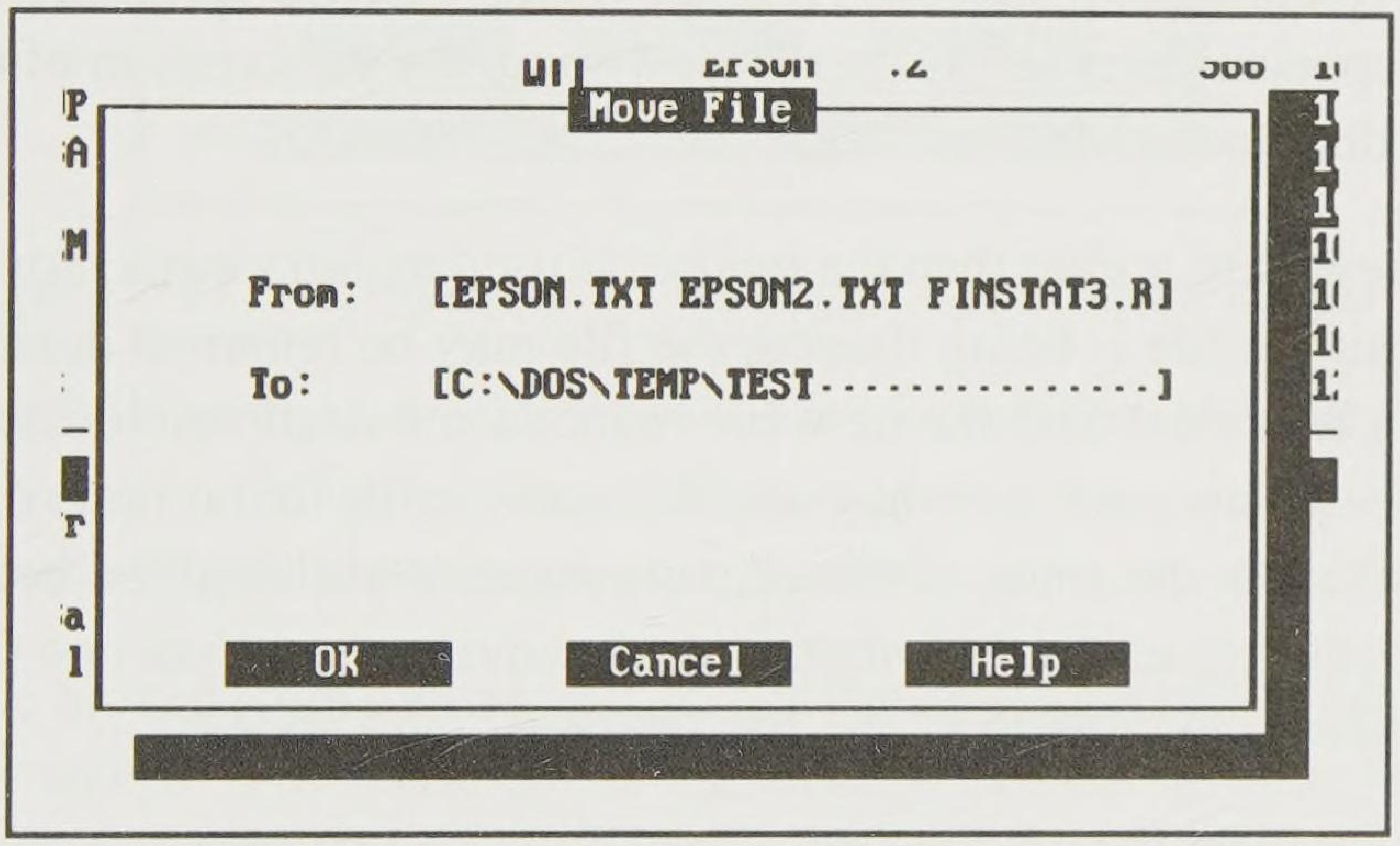
Are you sure you want to move the selected files to C:\DOS\TEMP?

Figure 2-25 — Dialog Box for Moving Files with a Mouse.

Selecting the Yes button will complete the move.

When moving files with the mouse, you should make sure the Directory Tree is expanded to show the destination directory, before starting the move operation. You may use the scroll bar to scroll through the Directory Tree, but you cannot expand the Directory Tree, during a move operation.

To move files using the keyboard, first select the files to be moved. Then press alt-F, to pull down the File menu. Then press M to select the Move option. Alternatively, after selecting the files to be moved, you can simply press F7, which is a shortcut for the Move option on the menu. A dialog box like that shown in Figure 2-26 will be displayed.



From: [EPSON.TXT EPSON2.TXT FINSTAT3.R]

To: [C:\DOS\TEMP\TEST................]

Figure 2-26 — Dialog Box for Moving Files with the Keyboard.

The cursor will be at the end of the **To:** text box. The selected files will appear in the From: text box. The Shift-tab combination moves the cursor into the **From:** text box. You cannot edit this text box, but you can move the cursor through the text in order to see file names that may be too long to appear in the space provided.

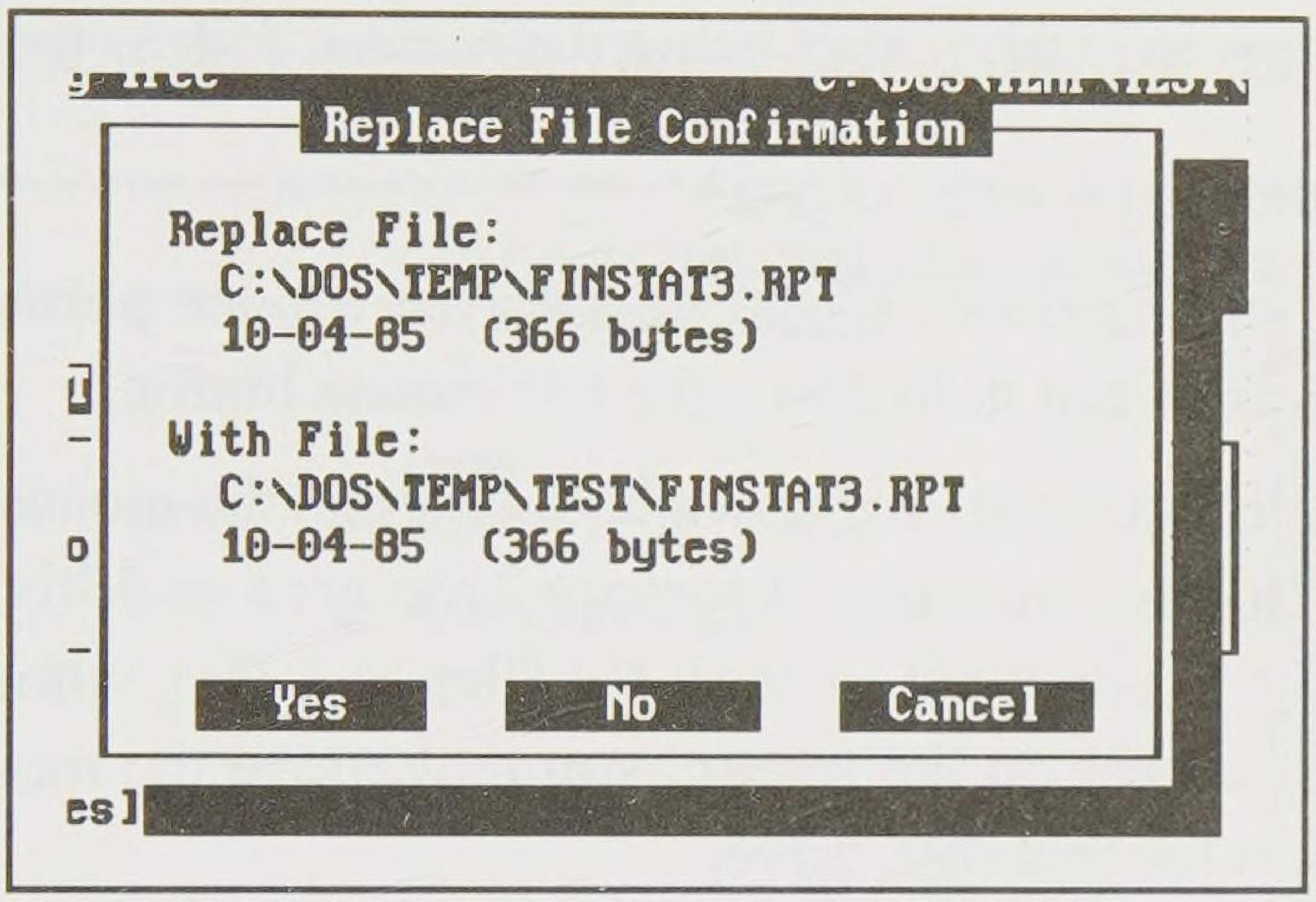
You may, however, edit the **To:** text box. The directory name shown here is the current directory, the one that files are being moved from. Since you cannot move files into the directory they are already in, you will need to edit this text box to specify the new directory. If you press any alphabetic key or ←BkSp, the current text disappears and whatever you type is entered on a blank line. If you press ←, the cursor jumps to the beginning of the text box, but the current text is left intact for editing. If you press →, the cursor remains in place, but anything you type is appended to the end of the existing text.

It is important to understand how DOS interprets a directory path that you include in a file name. If you specify a path whose final directory does not exist, that directory name will be interpreted as a file name. For example, if you enter the path C:\DOS\TEST and there is no TEST subdirectory in your DOS directory, DOS will interpret TEST as a new file name. This is because DOS has no way of differentiating between a file name and a directory name that comes at the end of a path. On the other hand, if one of the path's inner directories does not exist, then DOS knows that it is an invalid directory, since file names cannot occur in the inner part of a path. For example, if you enter C:\DOS\XX\TEST as the path name to move files to, DOS cannot tell whether TEST is a file or directory but knows that XX must be a directory. If XX does not exist the error message "Invalid Path" will appear. Refer to Chapter 5 for a complete explanation of the rules for specifying directory paths.

When the keyboard, rather than the mouse, is used to carry out a move operation, and only a single file is being moved, the file may be renamed during the move by adding a backslash and the new file name to the destination directory in the **To:** text box. In this case, you may actually move a file to the same directory it is in — the effect is the same as the rename operation (discussed below). If you attempt to specify a new name when you are moving more than one file, the error message "You have more than one file selected." will appear.

If the message "You have more than one file selected." appears and you were not attempting to enter a new file name, check the name of the directory you have entered to make sure it exists. If the last subdirectory in the path is not a directory, it will be interpreted as a file name that should be used for renaming the files being moved. For example, if you entered the target directory as C:\DS by mistake, then DS would be interpreted as a name that the selected files should be renamed to. Since you cannot rename multiple files to the same name the error message appears.

Moving a file to a directory that already contains a file of the same name causes the dialog box in Figure 2-27 to be displayed.


Replace File Confirmation

```
Replace File:
  C:\DOS\TEMP\FINSTAT3.RPT
  10-04-85  (366 bytes)

With File:
  C:\DOS\TEMP\TEST\FINSTAT3.RPT
  10-04-85  (366 bytes)
```

Figure 2-27 — Confirmation Dialog Box for Replacing a File.

Replacing one file with another destroys the one being replaced; this confirmation box allows you to review the status of the two files before continuing. Selecting the Yes button will complete the move operation.

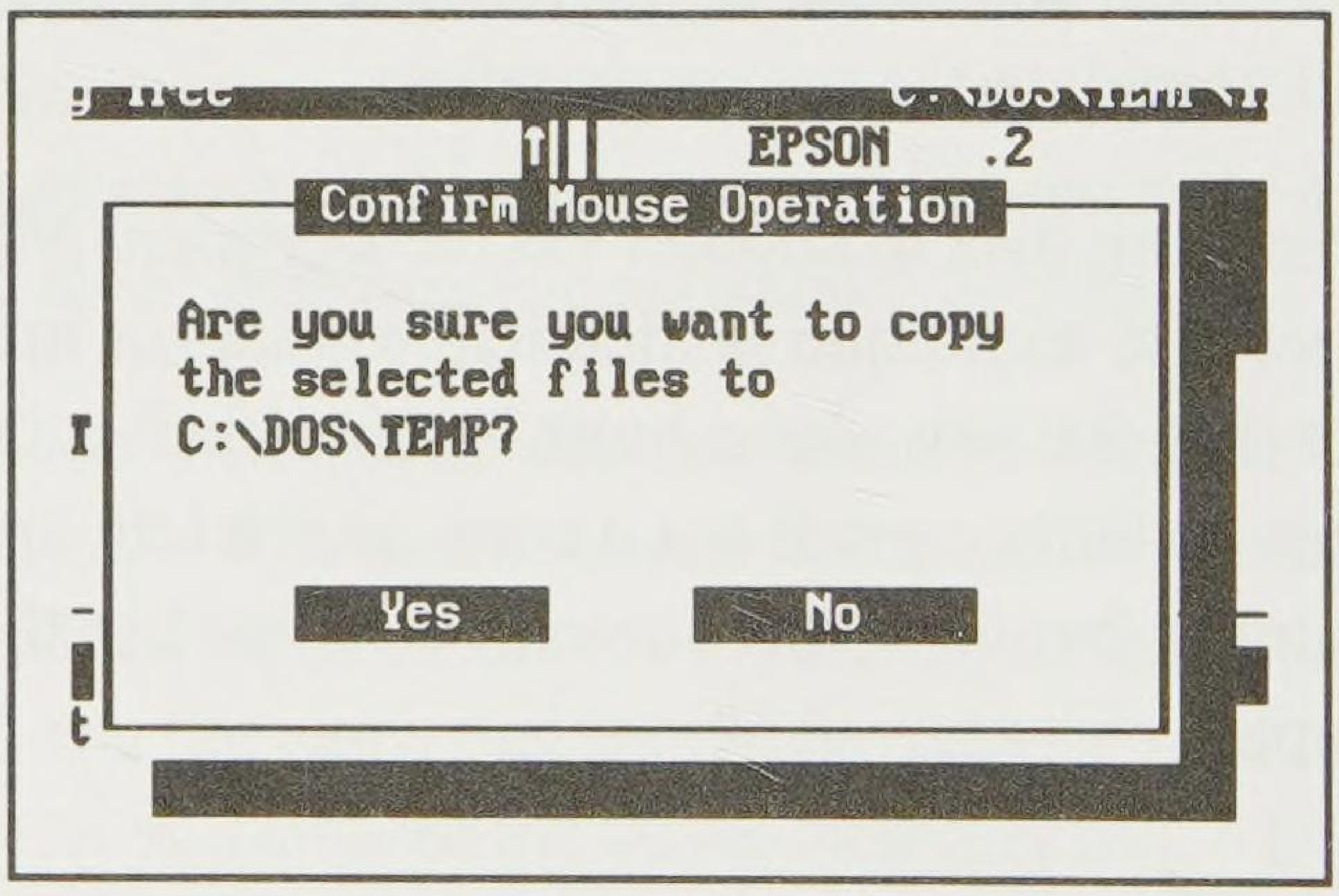
Files may be moved from disk to another via the keyboard MOVE operation, but not with the mouse. The technique is identical to that for moving files between directories, except that the new drive must be specified at the beginning of the directory path. This actually carries out a copy and delete operation, rather than a move. The difference between these two is discussed in the Advanced Topics section of this chapter.

Copying Files

Files may be easily copied between directories and disks within the Shell. To copy files from one directory to another using the mouse, follow these steps:

1. Select the file(s) to be copied
2. While holding down Ctrl, position the mouse pointer on one of the selected files and hold down the left mouse button.
3. While continuing to hold down both Ctrl and the mouse button, move the mouse pointer into the Directory Tree area and highlight the target directory. If you want to copy the files to a disk whose Directory Tree does not appear on the screen, you may move the mouse pointer to the drive icon for that disk drive.
4. Release Ctrl and the mouse button.

A dialog box like that in Figure 2-28 will appear.



Are you sure you want to copy the selected files to C:\DOS\TEMP?

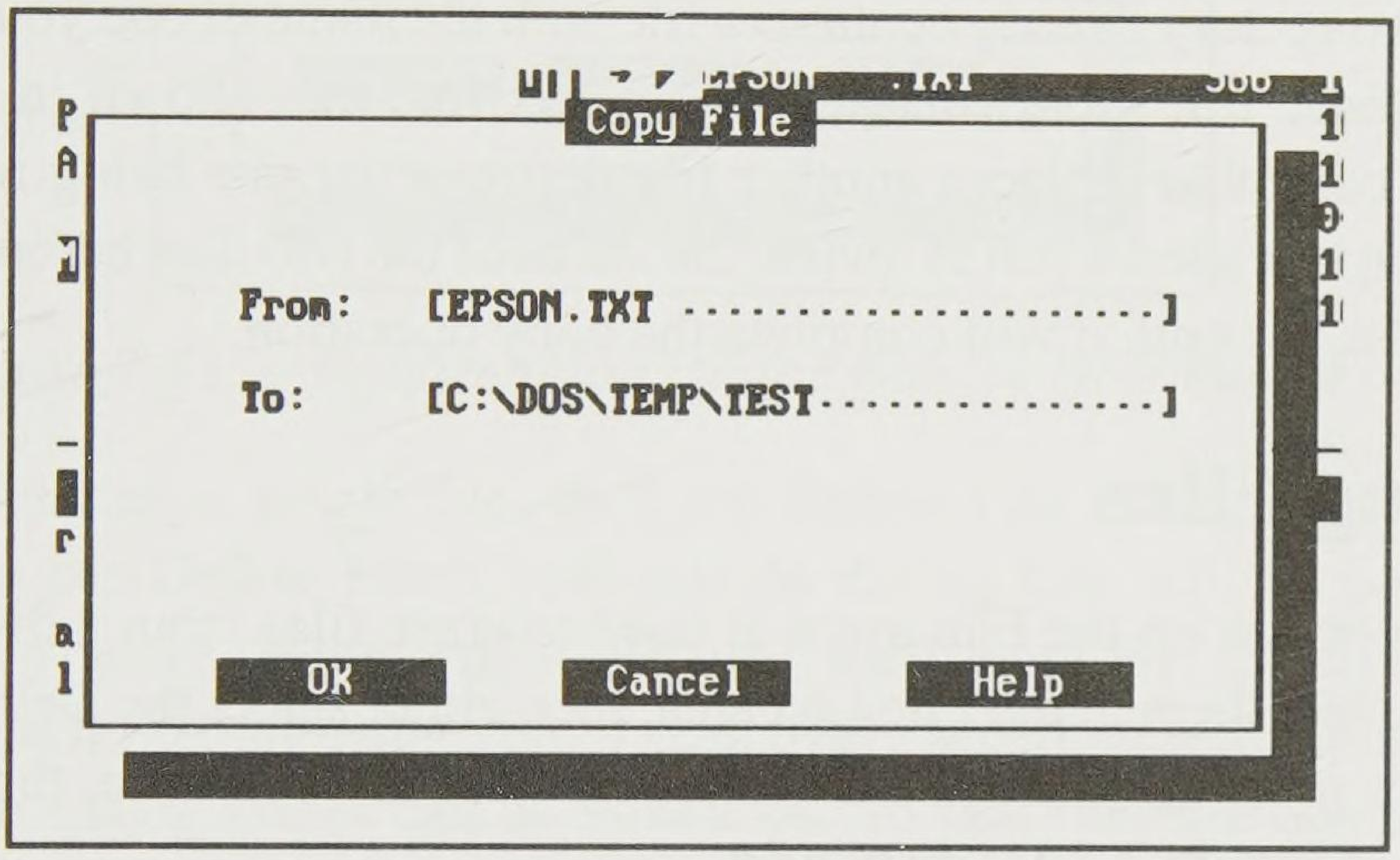
Figure 2-28 — Dialog Box for Copying Files with a Mouse.

Selecting the Yes button will complete the copy.

STOP

When copying files with the mouse, you should make sure the Directory Tree is expanded to show the destination directory, before starting the copy operation. You may use the scroll bar to scroll through the Directory Tree, but you cannot expand the Directory Tree, during a copy operation.

To copy files with the keyboard, select the files to be copied, press alt-F to pull down the File menu, then press C to select the Copy option. Alternatively, after selecting the files to be copied, you can press F8. This is the shortcut for the Copy option on the menu. A dialog box like that shown in Figure 2-29 will appear.



From: [EPSON.TXT ....]

To: [C:\DOS\TEMP\TEST.................]

Figure 2-29 — Dialog Box for Copying Files with the Keyboard.

The cursor will be at the end of the **To:** text box. The selected files will appear in the **From:** text box. You may only edit the **To:** text box. The directory name shown here when the dialog box comes up is the current directory, the one that files are being copied from. Since you cannot copy files into the directory they are already in without renaming them, you will need to edit this text box to specify the new file name or directory. If you press any alphabetic key, or ←BkSp, the current text disappears and whatever you type is entered on a blank line. If you press ←, the cursor jumps to the beginning of the text box but the current text is left intact for editing. If you press →, the cursor remains in place and anything you type is appended to the end of the existing text.

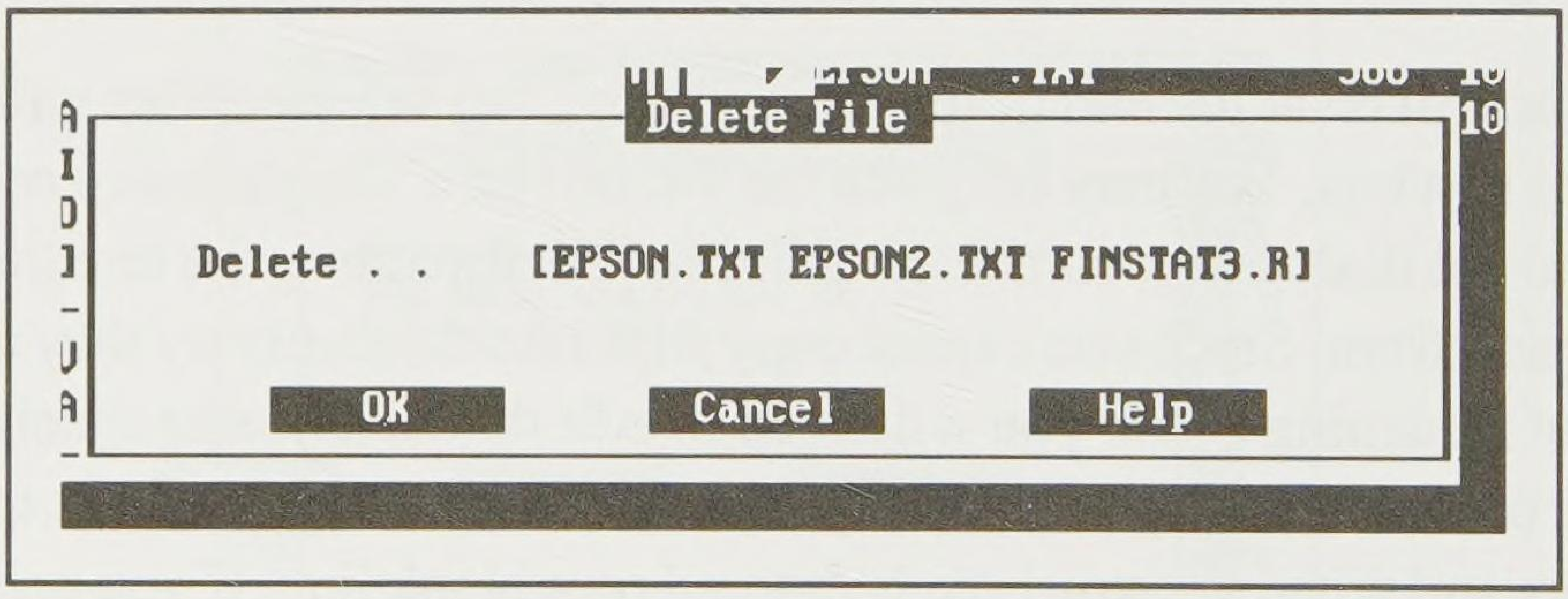
The ⇧Shift-tab combination moves the cursor into the **From:** text box. You cannot edit this text box, but you can move the cursor through the text in order to see file names that may not appear in the space provided.

When the keyboard, rather than the mouse, is used to carry out a copy operation, and only a single file is being copied, the file may be renamed during the copy by adding a backslash and the new file name to the destination directory in the **To:** text box. In this case, you may actually copy a file to the same directory it is in. There will now be two copies of the same file, with different names, in the same directory. If you attempt to specify a new name when you are copying more than one file, the error message "You have more than one file selected." will appear.

If the target directory already contains a file with the name of one you are copying to it, a Replace File Confirmation dialog box like that shown in Figure 2-27 appears. A copy that replaces another file destroys the one being replaced; this confirmation box allows you to review the status of the two files before continuing. Selecting the Yes button will complete the copy operation.

Deleting Files

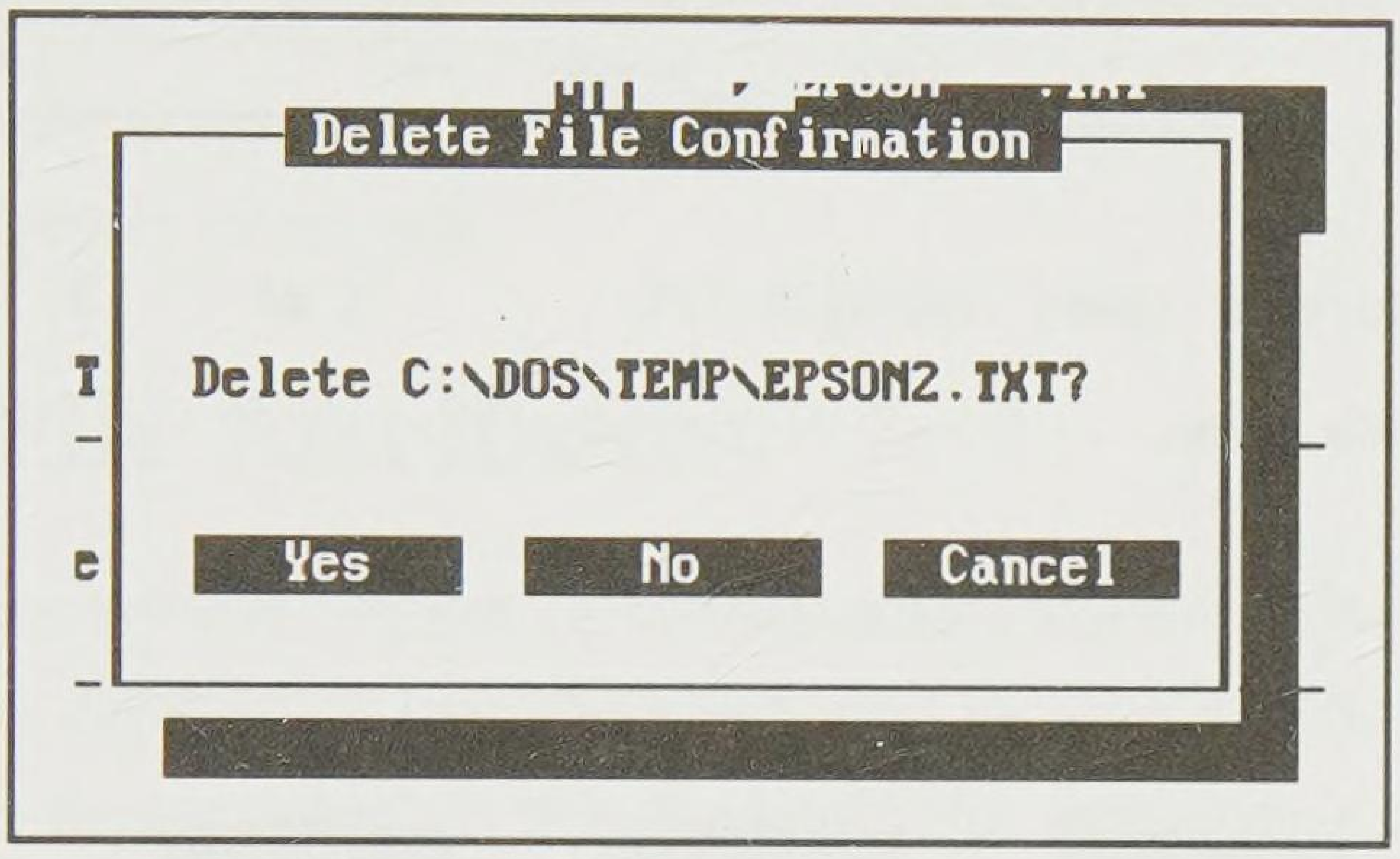
The Delete option on the File menu is used to erase files from a disk. To delete files, first select them in the File List area, then either select the Delete option on the menu or press delete. If you are deleting more than one file, the Delete File dialog box in Figure 2-30 is displayed.



Delete . .   [EPSON.TXT EPSON2.TXT FINSTAT3.R]

Figure 2-30 — Delete File Dialog Box.

This dialog box shows a list of all the files you are going to delete. The cursor is on the OK button, but you can use ⇧ Shift-tab to move it into the text box. You cannot edit the text box, but you can use the arrow keys to scroll through it, to see any files that don't fit in the available space. Selecting the OK button brings up the Delete File Confirmation dialog box shown in Figure 2-31.



Delete C:\DOS\TEMP\EPSON2.TXT?

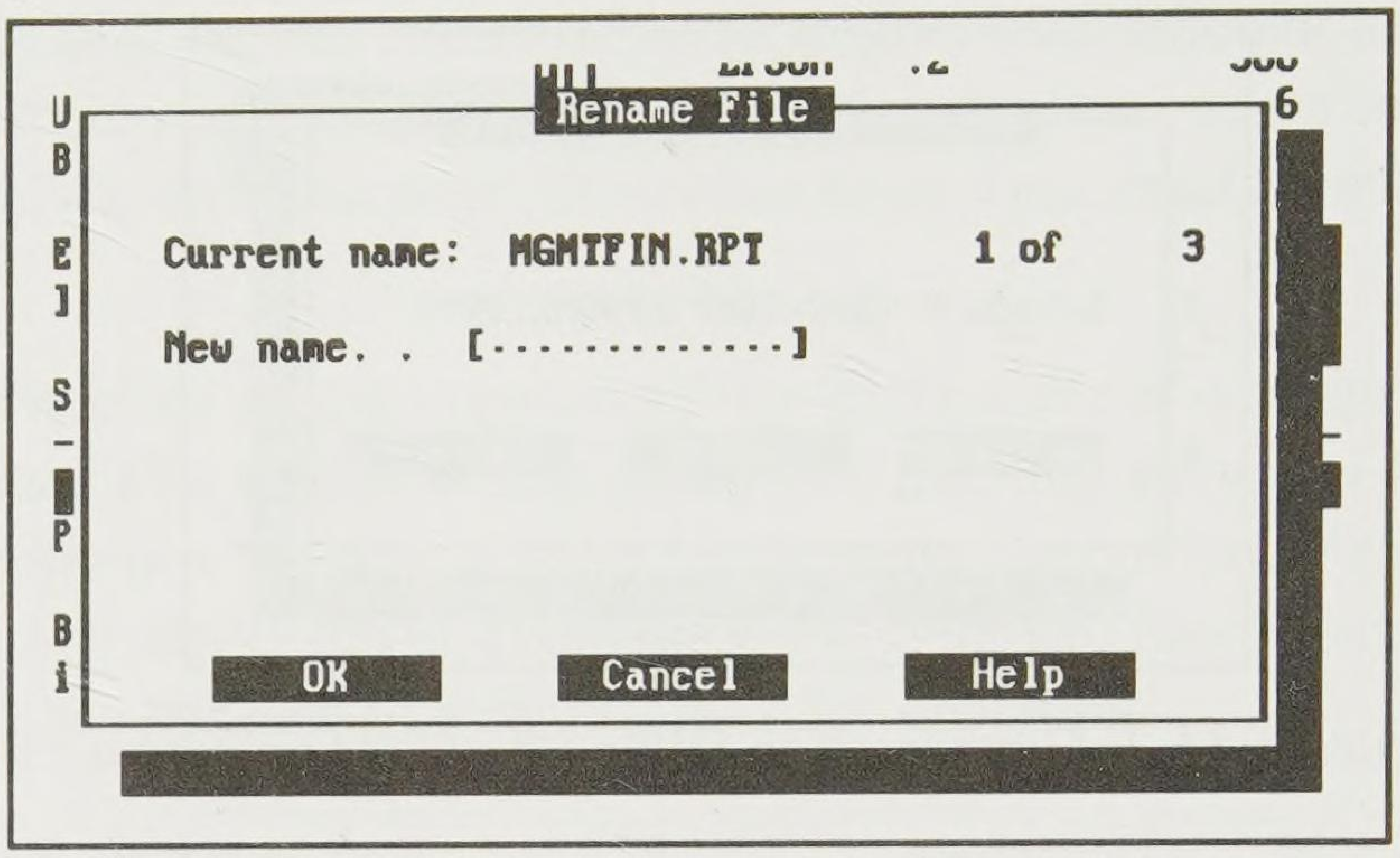
Figure 2-31 — Confirmation Dialog Box for Deleting a File.

If you are deleting a single file, then the Delete File dialog box will not be displayed — the Delete File Confirmation dialog box will be immediately displayed. If you are deleting multiple files, the Delete File Confirmation box appears for each file, allowing you to selectively accept or reject each deletion. Confirmation dialog boxes can be turned off, so that files are deleted without requiring your review. Turning off confirmation dialog boxes is discussed later in this chapter, in the section titled, “Managing the Shell Environment.”

It is sometimes possible to recover a file that has been accidentally deleted. See “Undeleting Files” later in this chapter and in Chapter 5, for directions on undeleting files. For the best chance of success, undeletion should be attempted as soon as possible after the deletion.

Renaming Files

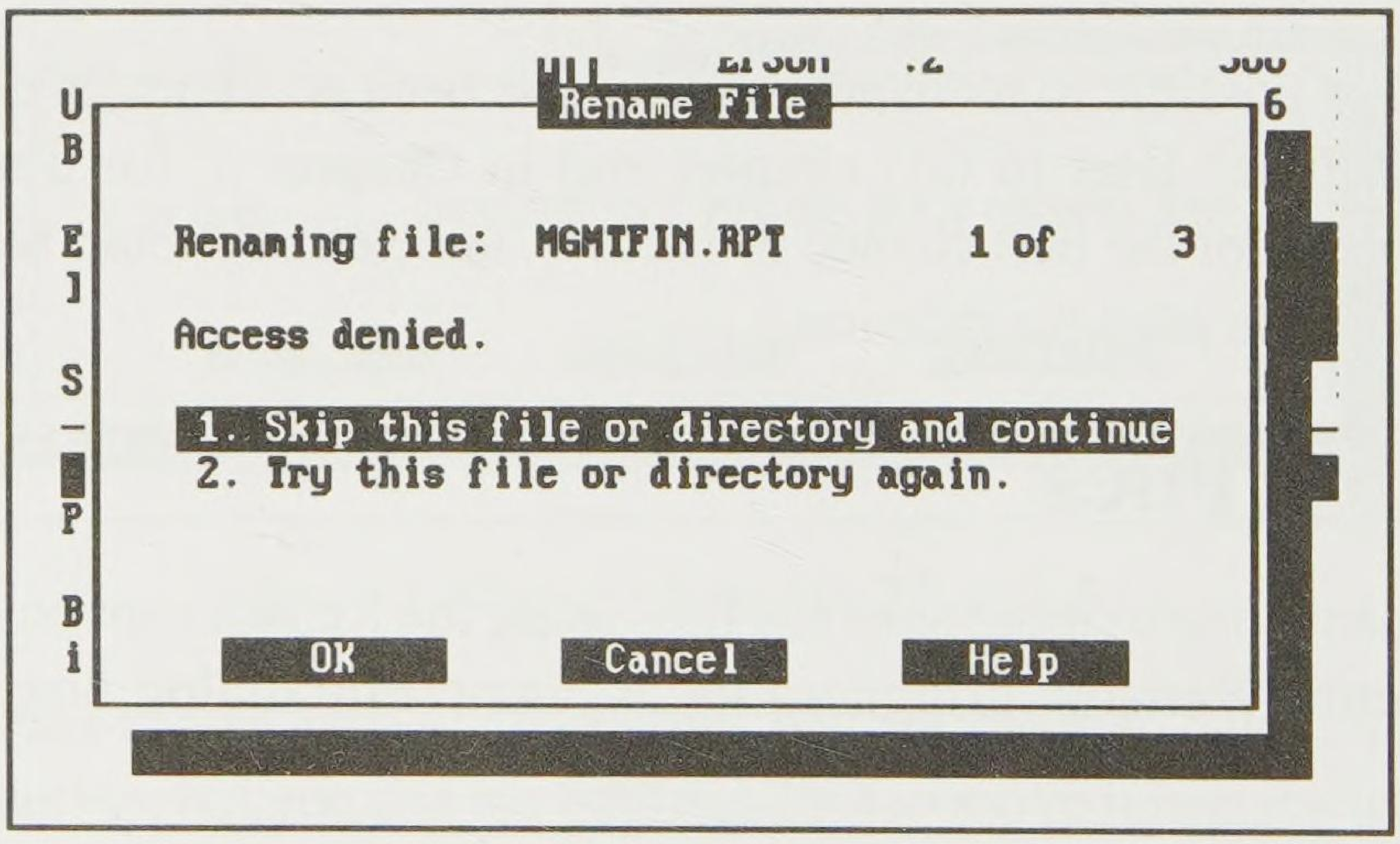
You may rename one or more selected files using the Rename option on the File menu. Selecting Rename brings up the Rename File dialog box, shown in Figure 2-32.



```
Current name:  MGMTFIN.RPT           1 of     3

New name. .  [............]
```

Figure 2-32 — Dialog Box for Renaming Files.

The "1 of 3" in the upper right corner indicates that three files were selected for renaming, and that this is the first of them. The new name for the file is entered in the New name text box. If you enter the name of a file that already exists, the Rename File dialog box changes to that shown in Figure 2-33.



```
Renaming file:  MGMTFIN.RPT          1 of     3

Access denied.

 1. Skip this file or directory and continue
 2. Try this file or directory again.
```

Figure 2-33 — Dialog Box when Attempting to Rename to a Duplicate File Name.

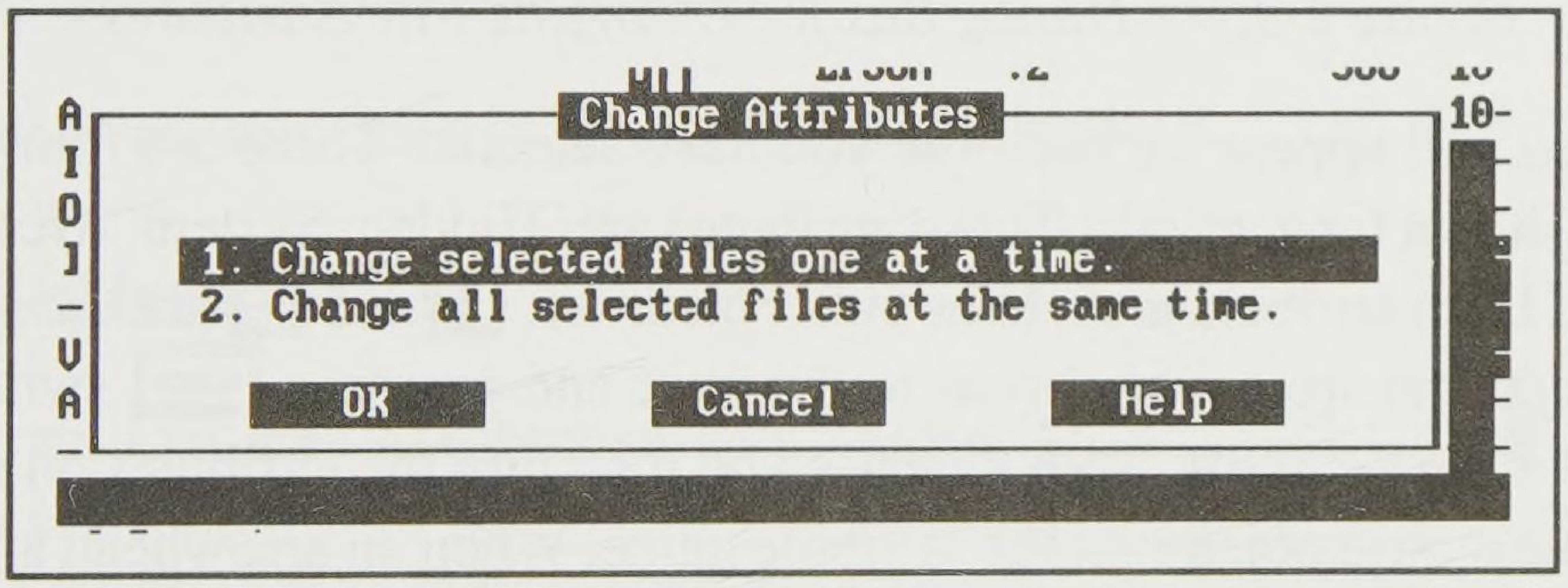
Selecting option 1 brings back the dialog box in Figure 2-32 for the next file in the list. Selecting option 2 brings back the dialog box in Figure 2-32 for the selected file.

When a file is renamed, the file itself is not touched. The name is stored in the directory, so the only change that occurs in a Rename operation is that the file's name in the directory is changed.

Changing File Attributes

The directory stores several items of information about a file: its file name, the date and time it was created or last changed, and its attribute settings. Under most circumstances you never need to think about the attribute settings, so they are not normally shown. However, they may be displayed and altered using the Change Attributes option on the File menu. You must first select one or more files before you can choose the Change Attributes option.

If you have selected more than one file, Change Attributes brings up the dialog box shown in Figure 2-34.



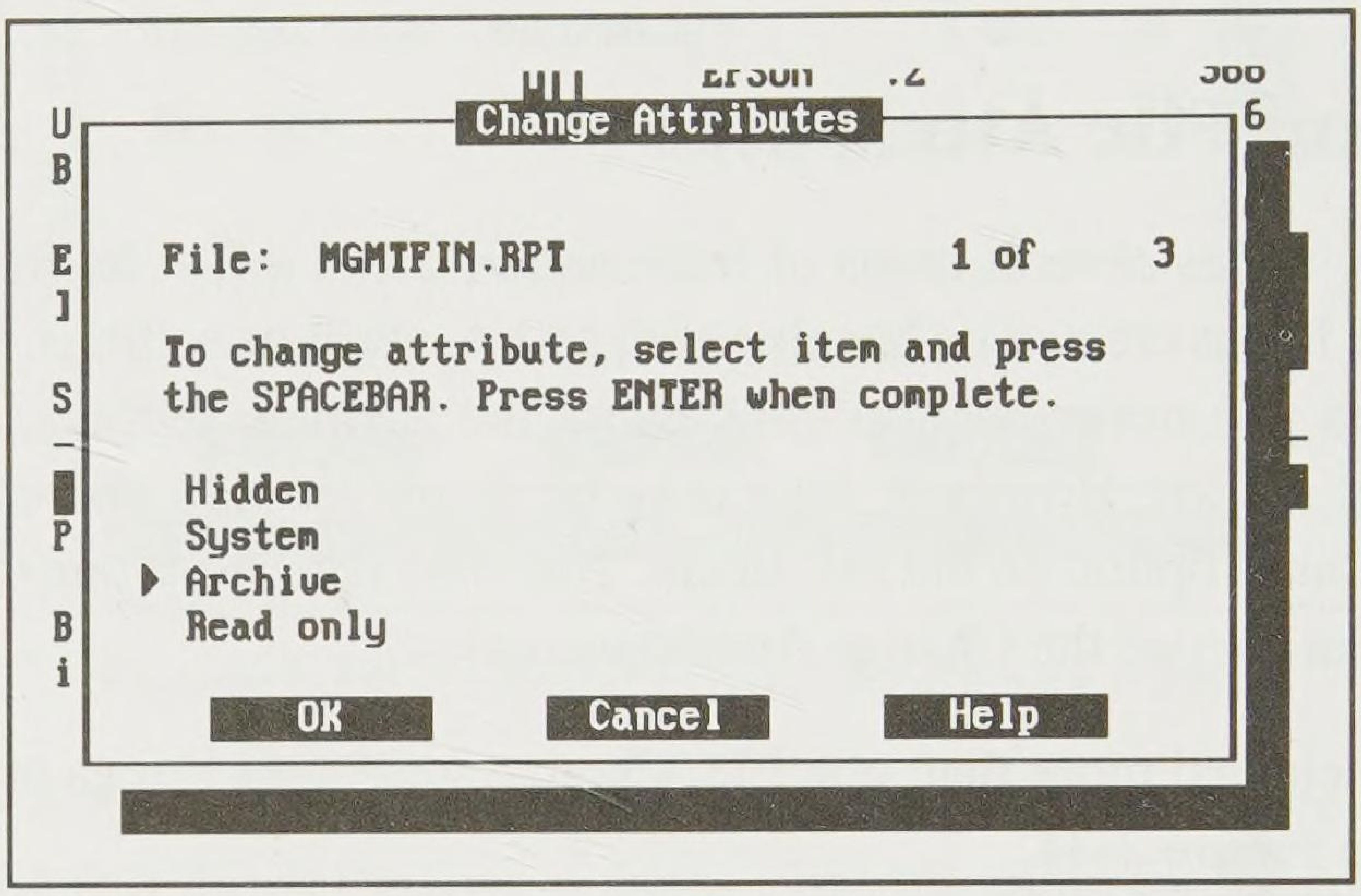
1. Change selected files one at a time.
2. Change all selected files at the same time.

Figure 2-34 — Dialog Box for Confirming File Attribute Changes.

This dialog box gives you the option of reviewing each file before the attributes are changed, or changing the attributes for all selected files at once. Option 2 (Change all selected files at once) should be used with caution; inadvertently changing a file's attributes can result in a variety of problems, including disabling programs and leaving vital files vulnerable to accidental loss.

Choosing option 1 (Change selected files one at a time) will bring up the dialog box in Figure 2-35.

If only one file is selected in the File List, then Figure 2-35 is the first dialog box you will see.



Change Attributes

File: MGMTFIN.RPT 1 of 3

To change attribute, select item and press the SPACEBAR. Press ENTER when complete.

Hidden
System
▶ Archive
Read only

Figure 2-35 — Dialog Box for Changing File Attributes.

This screen will appear for each file you have selected. There are four kinds of attributes shown for each file. These attributes are: Hidden, System, Archive, and Read only. Each attribute is set to be either on or off. ↑ and ↓ are used to move the highlight bar from one choice to the next and pressing space switches the attribute setting on or off. With a mouse you may turn the attributes on or off by clicking the mouse pointer on the attribute name. When an arrowhead appears to the left of a setting it indicates that the setting in on. In Figure 2-35, the only setting that is on is the Archive setting.

The **Hidden** attribute is used to hide a file. When Hidden is on, the file does not display when you perform a directory listing at the command prompt (the DIR command), or if you list a directory from software within packages such as word processors. Hidden files will show up in the File List of the Shell if the "Display hidden/system files" box is checked in the File Display Options dialog box (see Figure 2-12). Hidden files are sometimes used for certain types of copy protection schemes, and for certain system files required by DOS.

The **System** attribute indicates that a file is used by DOS for operating system functions. There are two hidden system files that come with DOS, called IO.SYS and MSDOS.SYS.

The **Archive** attribute indicates whether a file has been changed since it was last backed up. It is used by the DOS Backup, Restore, and XCopy commands, as well as disk backup software that can be purchased from software dealers. When a file has been backed up the Archive attribute is set to OFF. As soon as the file has been modified in any way, DOS sets the Archive attribute to ON. This allows you the option to perform backups only on files that have been changed since the previous backup.

The **Read-only** attribute determines whether a file can be read and written to, or only read. This is useful to protect important files from being deleted or changed. If the read attribute is on, the file can only be read. If the read attribute is off the file can be read, written to, and deleted.

File attributes are discussed in greater detail in Chapter 5.

Undeleting Files

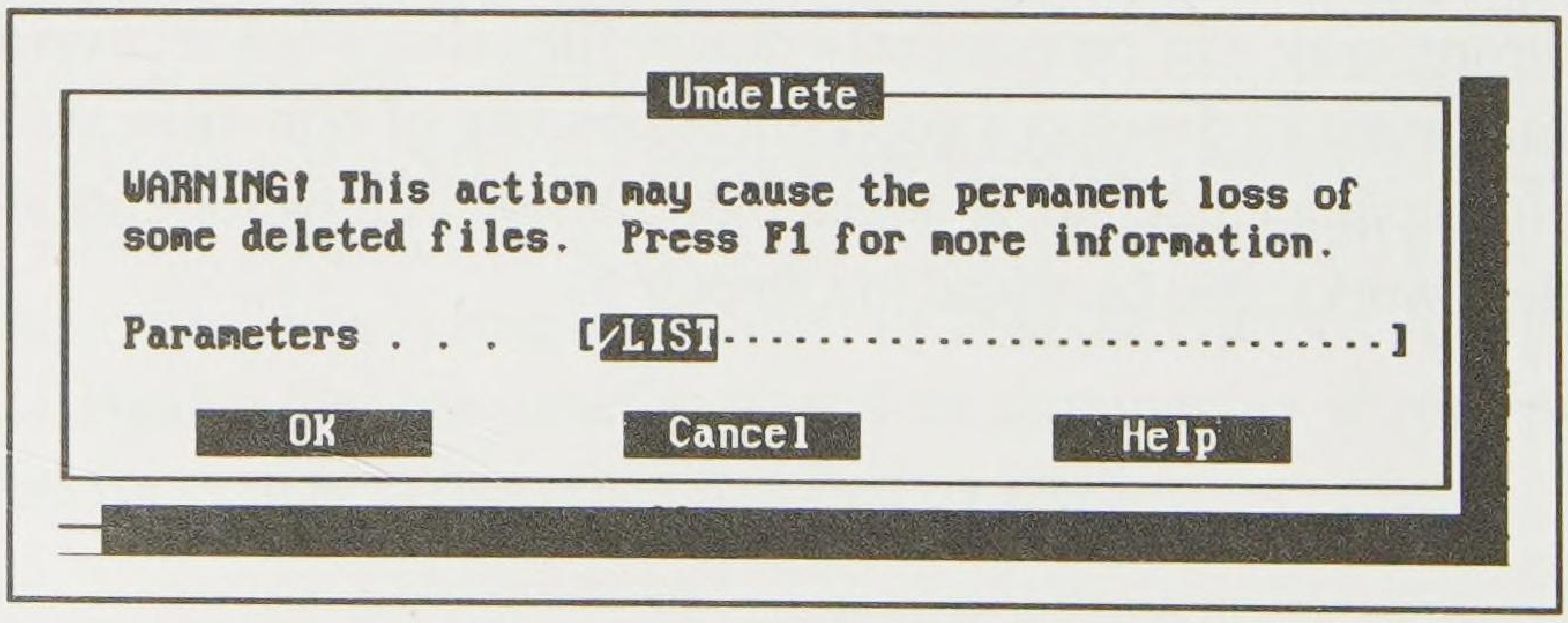
The Program List area has a menu item named [Disk Utilities]. Selecting this brings up a second menu, which contains several DOS programs for managing your disks. The last item on this menu, Undelete, is used for recovering files that have been accidentally erased.

Undelete does not always work. In fact, using it under the wrong circumstances or in the wrong way can permanently delete files that were recoverable. It is therefore important to develop a good understanding of how files are stored on the disk before using Undelete. A detailed discussion of how files are stored, and how Undelete works, can be found in Chapter 5.

For now, there are a few points that are important to know:

1. A deleted file should be recovered as soon as possible after it has been deleted. The more activity the computer carries out after a file is deleted, the less likely it is to be recoverable.
2. Undelete should NEVER be used when the Task Swapper is active.
3. You should not attempt to undelete a file on a full, or nearly full disk.
4. Directories cannot be undeleted, only files. If a directory whose parent is the root directory has been deleted, it may be possible to recover that directory using the UNFORMAT command as discussed in Chapter 5. Directories whose parent is not the root directory cannot be recovered.
5. Once a directory is deleted, the files that were in it can only be undeleted if the directory itself can first be recovered using UNFORMAT.
6. There is another DOS program, discussed in Chapter 5, named MIRROR. If MIRROR is loaded into memory, it maintains a file of detailed information about all files that are deleted on the disk. This improves the likelihood of completely and successfully recovering deleted files. If MIRROR was not loaded when the file was deleted, Undelete will attempt to recover the file from information that may still exist in the directory.

To undelete one or more files, first select the directory that contained them. If the files you want to undelete were in more than one directory, you will have to run Undelete for each directory. Selecting Undelete brings up the dialog box shown in Figure 2-36.



```
WARNING! This action may cause the permanent loss of
some deleted files.  Press F1 for more information.

Parameters . . .   [/LIST............................]
```

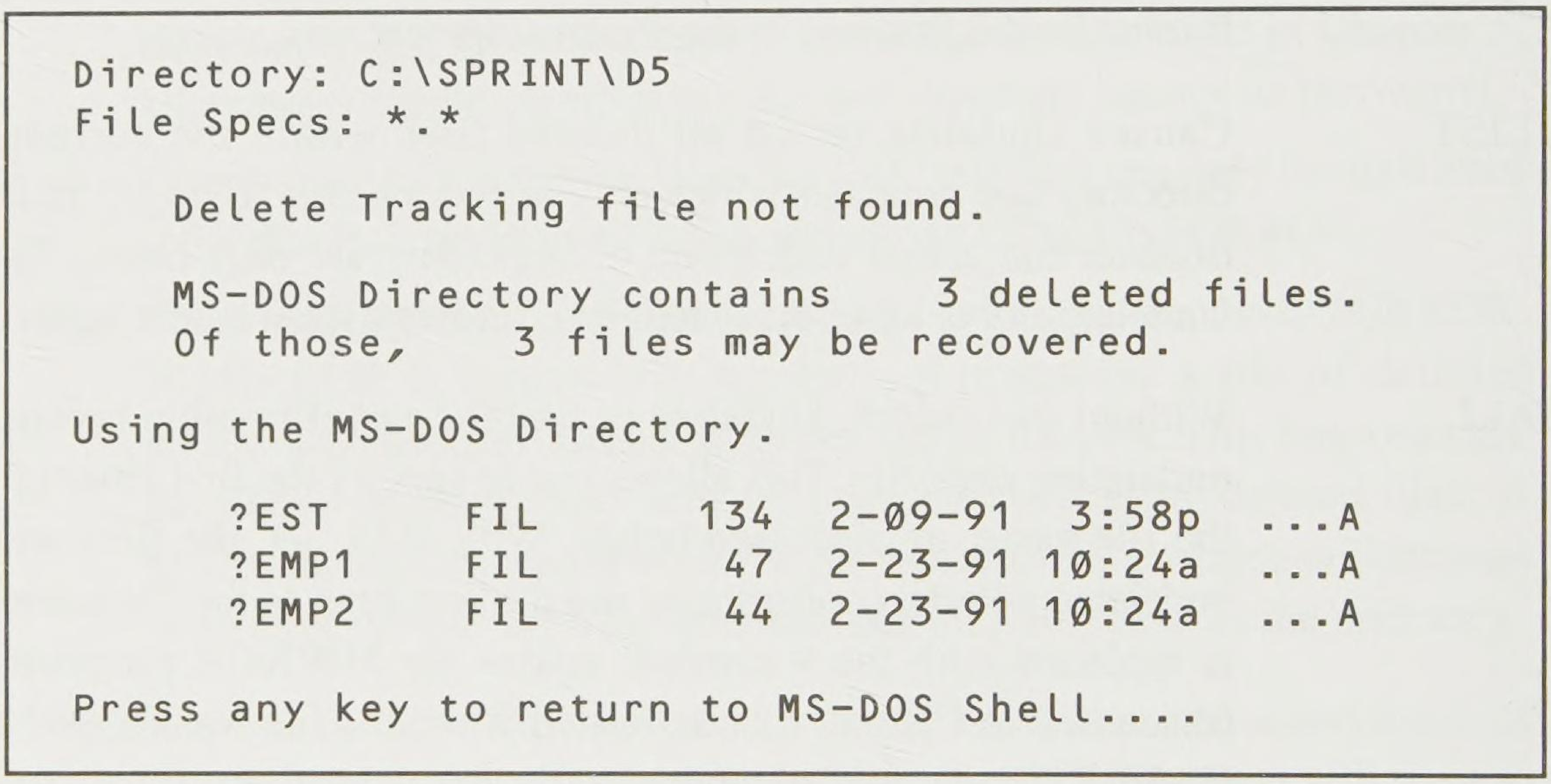
Figure 2-36 — The Undelete Dialog Box.

When entered on the command line, there are several optional parameters that may be added to the Undelete command. The Parameters text box in Figure 2-36 can be used to enter those parameters when running Undelete from the Shell. Parameters may be entered in either upper or lowercase — Undelete is not case sensitive. The possible parameters and their effects are:

D:\DIR\FILE	Specifies a drive, path, and file specifier (either a specific file name or a file name with wild cards to select a group of files). This overrides the currently selected directory, so that the command acts on the specified directory. If this parameter is used, it must be the first one in the Parameters text box.
/LIST	Causes Undelete to list all deleted files within the current directory and indicates which may be recoverable. Using /LIST disables the actual undeletion of files, they are only listed. To Undelete any or all of the listed files, Undelete must be run again.
/ALL	Without this switch, Undelete pauses for a confirmation before undeleting each file. This allows you to specify the first letter of the file name, as discussed below. With /ALL set, the files are undeleted without confirmation and the first letter of the file name is replaced with the # symbol, unless the MIRROR program (discussed in Chapter 5) was loaded when the file was deleted. If MIRROR was loaded when the file was deleted, the file name will be fully restored.
/DOS	If the MIRROR program was loaded when a file was deleted, then Undelete normally uses the information saved by MIRROR to restore the file. This sometimes makes it possible to recover files that could not be recovered without the MIRROR information. However, if /DOS is specified, the MIRROR information is ignored. The details of how this works are discussed in Chapter 5.
/DT	This parameter is the opposite of /DOS. It causes Undelete to only recover files for which there is MIRROR information available. You are prompted for confirmation before recovering each file it finds.

You do not have to enter anything in the Parameters text box. If you delete the default /LIST parameter, and do not enter anything else, Undelete goes ahead and attempts to recover each file it finds in the currently selected directory. You will be prompted for confirmation on each file before it is recovered, allowing you to skip files you do not want to recover.

For example, if three files named TEMP1.FIL, TEMP2.FIL, and TEST.FIL were deleted from the C:\SPRINT\D5 directory, then in response to the dialog box in Figure 2-36, pressing Enter← will display the screen shown in Figure 2-37.

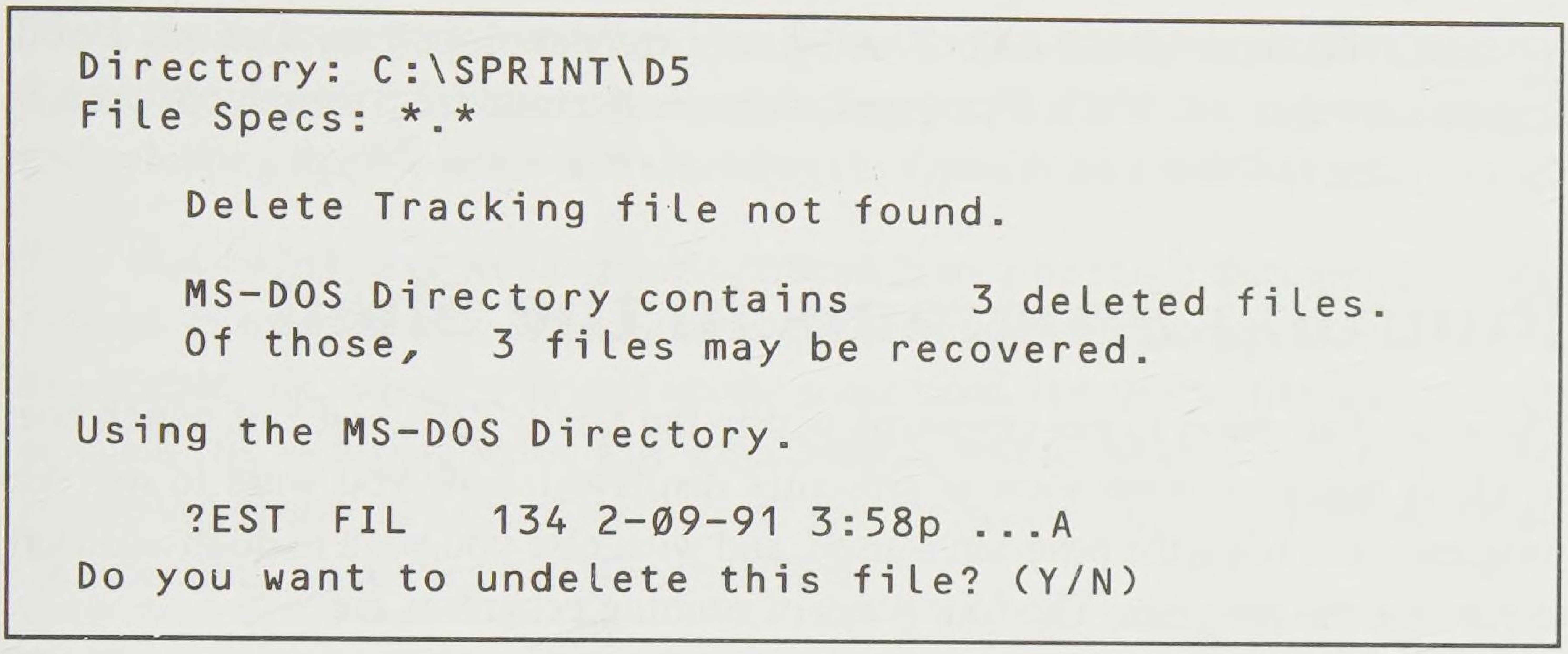


```
Directory: C:\SPRINT\D5
File Specs: *.*

    Delete Tracking file not found.

    MS-DOS Directory contains    3 deleted files.
    Of those,    3 files may be recovered.

Using the MS-DOS Directory.

      ?EST      FIL      134  2-09-91   3:58p  ...A
      ?EMP1     FIL       47  2-23-91  10:24a  ...A
      ?EMP2     FIL       44  2-23-91  10:24a  ...A

Press any key to return to MS-DOS Shell....
```

Figure 2-37 — Results of Undelete Option with /LIST Parameter.

The message “Delete Tracking file not found.” indicates that the MIRROR program was not loaded when the files were deleted. The three files that had been deleted were all located by Undelete. The first character of each file name is missing. This is how DOS deletes files — by marking the first character of the file name in the directory with a special code that means the file has been deleted. That is ALL DOS does when it deletes a file. The file information itself is not changed, which is why it is possible to undelete a file. Pressing any key returns you to the Shell.

Now that the recoverable files have been listed, to actually recover them, Undelete must be selected again, this time without the /LIST parameter. (You do not have to run Undelete with /LIST first, but usually it is helpful to do so. Running Undelete with no parameters brings up the screen shown in Figure 2-38.



```
Directory: C:\SPRINT\D5
File Specs: *.*

    Delete Tracking file not found.

    MS-DOS Directory contains    3 deleted files.
    Of those,   3 files may be recovered.

Using the MS-DOS Directory.

    ?EST  FIL    134 2-09-91 3:58p ...A
Do you want to undelete this file? (Y/N)
```

Figure 2-38 — Results of Undelete Option with No Parameters Specified.

When Y is pressed, the following prompt appears:

```
Enter the first character of the filename.
```

Entering a letter produces the following message:

```
File successfully undeleted.
```

The next deleted file name, with a question mark for the first letter, is then displayed as above, and you are again asked if you want to undelete the file. This process continues until all files have been processed and you are prompted to

```
Press any key to return to MS-DOS Shell...
```

Under the best of circumstances — when the file is relatively small, the disk is relatively unfragmented, and no disk activity has occurred since the file was deleted — recovering a file is straightforward and safe. However, if these circumstances are not all met, undeletion can become a complex task with many hidden dangers. It is important to have a good understanding of how files are managed in order to make the best use of the Undelete program. There are circumstances under which Undelete cannot recover a file, but other programs can. These programs are specialized utilities that have been developed for disk repair and file recovery; some of them seem almost capable of miracles. If the deleted or damaged file is very important, it may be worth the time and money to purchase one of these programs and to learn how to use it. Some good examples are Mace Utilities, Norton Utilities, and PC Tools. (In fact, the MIRROR, UNDELETE, and UNFORMAT utilities were developed by Central Point Software for their PC TOOLS program, and were licensed from them by Microsoft for inclusion in DOS 5.0)

Running Programs in the Shell

There are four ways to run programs within the Shell. The choice of which one to use is based on what view is presently displayed, how you want to use the program, how often the program is used, and what else you want to do in addition to running the program. The four ways of running programs are:

1. When a program file, or a file associated with a program, has been selected in the File List area, the program may be run by pressing Enter↵, by clicking twice with the mouse pointer on the file name, or by choosing Open from the File menu.
2. Run may be selected from the File menu. This provides you with a dialog box in which you may type the name of the program you want to run.
3. Programs may be run from the command prompt. To go to the command prompt, either select Command Prompt in the Main group of the Program List area, or press ⇧Shift-F9. The advantages of the Run option are that a little more memory is available for the programs, and that it is not necessary to select the Run option for each command you want to enter.
4. A program can be added to a Program List group, then selected from the group menu. You must first select the group if it is not already displayed.

... from the File List Area

Program file names must have one of the following three extensions: BAT, COM, or EXE. It is these extensions that tell DOS that a file is an executable program. Each extension represents a different kind of executable file, and when DOS runs a program it does so in a way determined by the extension. The three extensions are not interchangeable — you cannot rename a .COM file to an .EXE file, for example. Files with any one of these three extensions may be run from the File List area.

There are three ways to run a program from the File List area: clicking twice on the file name with the mouse pointer; selecting the file name and pressing Enter↵; and selecting the file name and choosing the Open from the File menu.

Files that do not have an executable extension may be run if they are associated with an executable file. Any one of these techniques can be used on this non-executable file, which will call up the associated executable file and pass the selected file as a parameter. File associations, and techniques for creating and removing them, are discussed in an earlier section of this chapter titled, "Associating Data Files with Program Files."

... from the RUN Option

The Run option on the File menu can be used to run programs in the same way you would run them from the command prompt, without exiting the Shell. When Run is selected, the dialog box shown in Figure 2-39 is displayed to prompt you for a command.

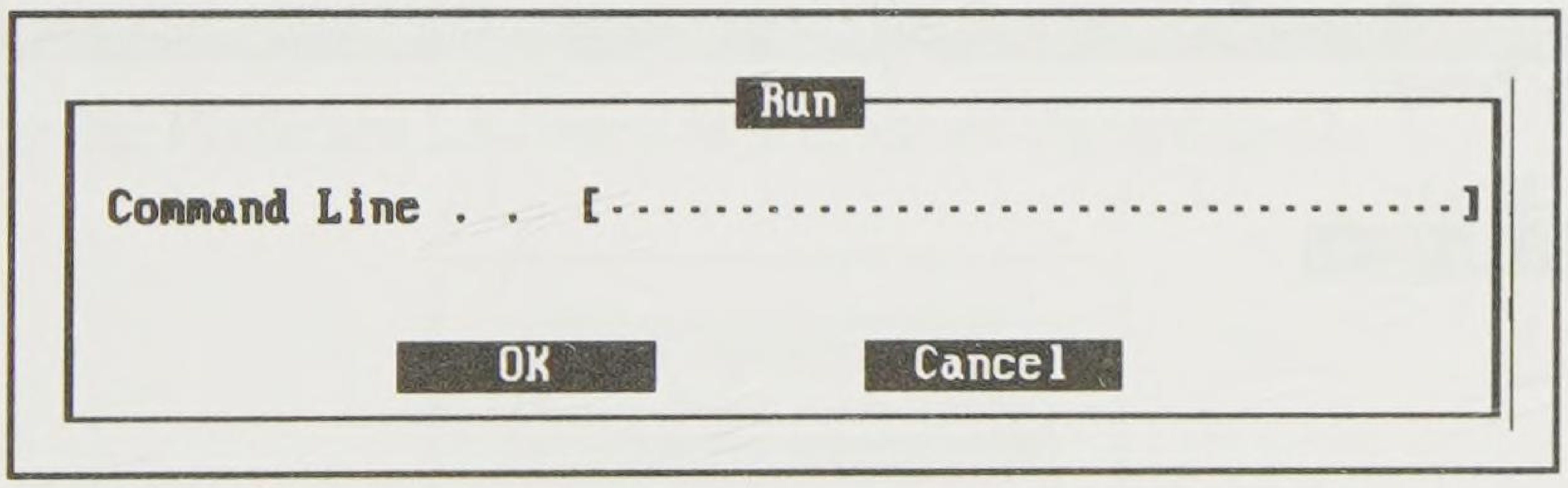


Figure 2-39 — The Run Dialog Box.

The command entered in the text box can be anything that could be entered at the DOS command prompt. This includes DOS commands as well as programs.

... from the Program List Area

The Program List area, at the bottom of the screen in Figure 2-1, provides you with a menu system for programs that you frequently use. Programs in the Program List area can have their own custom Help and startup parameter dialog boxes. There are several DOS programs already installed in the Program List. Others may be added whenever you like. In addition to adding programs, you may add more menus. Adding new programs, menus, Help boxes, and Parameter dialog boxes, is discussed later in this chapter, in the section titled, "Managing the Program List Area."

There are three ways to run a program or select a new group in the Program List area: you may select the item and press Enter←, you may select the item and choose Open from the File menu, or you may double click the mouse pointer on the item.

Managing the Program List Area

The view can be changed to display only the Program List by selecting Program List from the View menu. This makes it possible to see long menus without having to scroll through them. It also makes it easier to focus on just the Program List. The Program List view is shown in Figure 2-40.

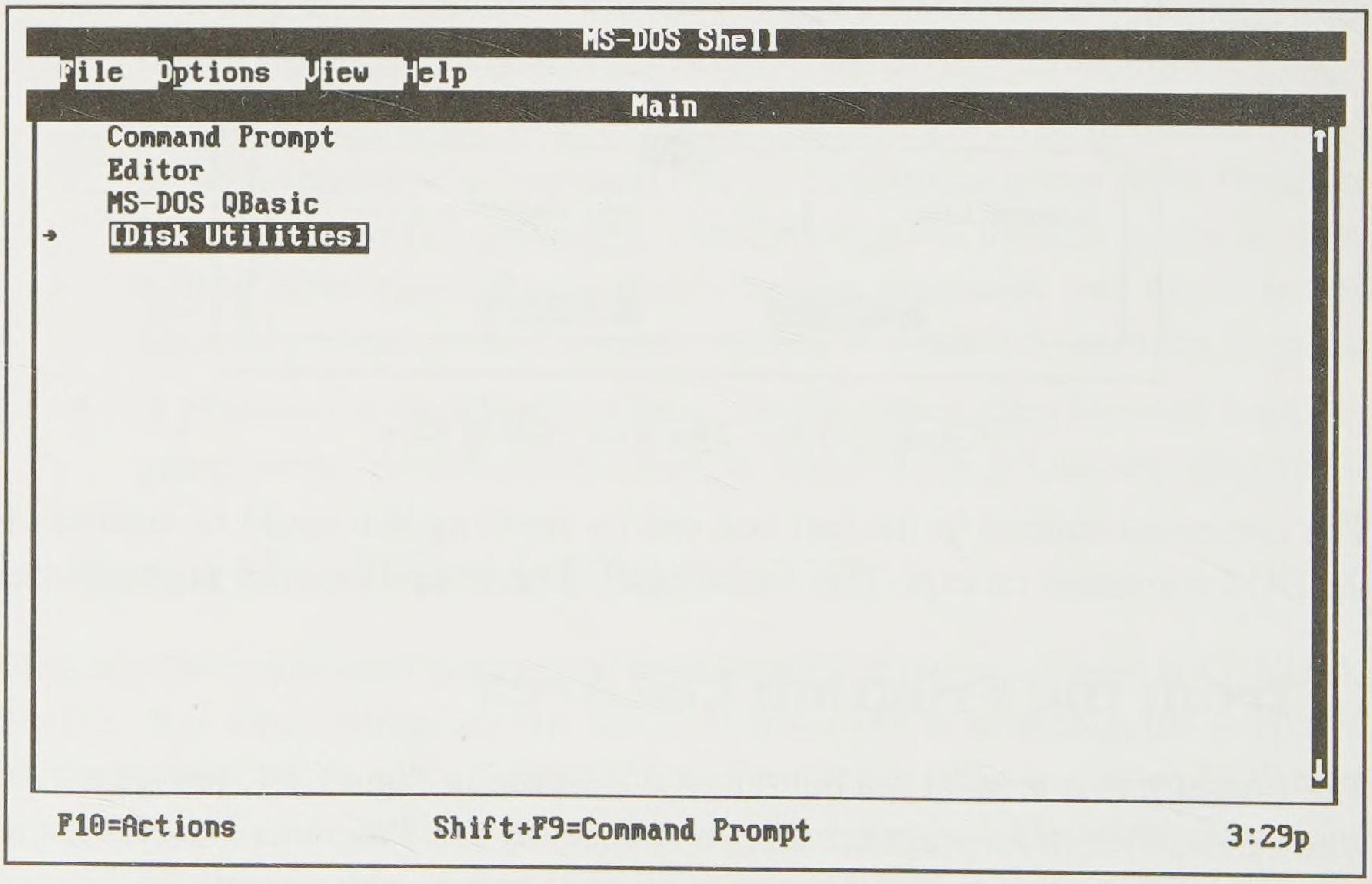

MS-DOS Shell
File Options View Help
Main
Command Prompt
Editor
MS-DOS QBasic
[Disk Utilities]
F10=Actions
Shift+F9=Command Prompt
3:29p

Figure 2-40 — Program List View.

In the Program List area, a menu consists of a group of one or more items. There are two kinds of items that can be included in a menu group: the name of a program, and the title of another group. In Figure 2-40, there are three programs and one group title. The programs are Command Prompt, Editor, and MS-DOS QBasic; the group is [Disk Utilities]. Groups are identified by the brackets that surround their names. The brackets are automatically added by the Shell, so you don't need to type them when entering the group name.

Menu items are selected in the usual ways — pressing the first letter of the item name, using the arrow keys, or using the mouse pointer. You can also use home and end to go to the top or bottom of a list. When a program is selected, it can be run. When a group title is selected, the menu for that group can be displayed. Help dialog boxes can be defined for both programs and groups. Passwords can also be defined for programs and groups, allowing you to restrict access to them. Parameter boxes can be defined for programs (these are used to pass parameters to programs when they start up).

When the Program List area is active, the File menu is used for managing the Program List rather than managing the File List and Directory Tree areas. The File menu for the Program List area is shown in Figure 2-41.

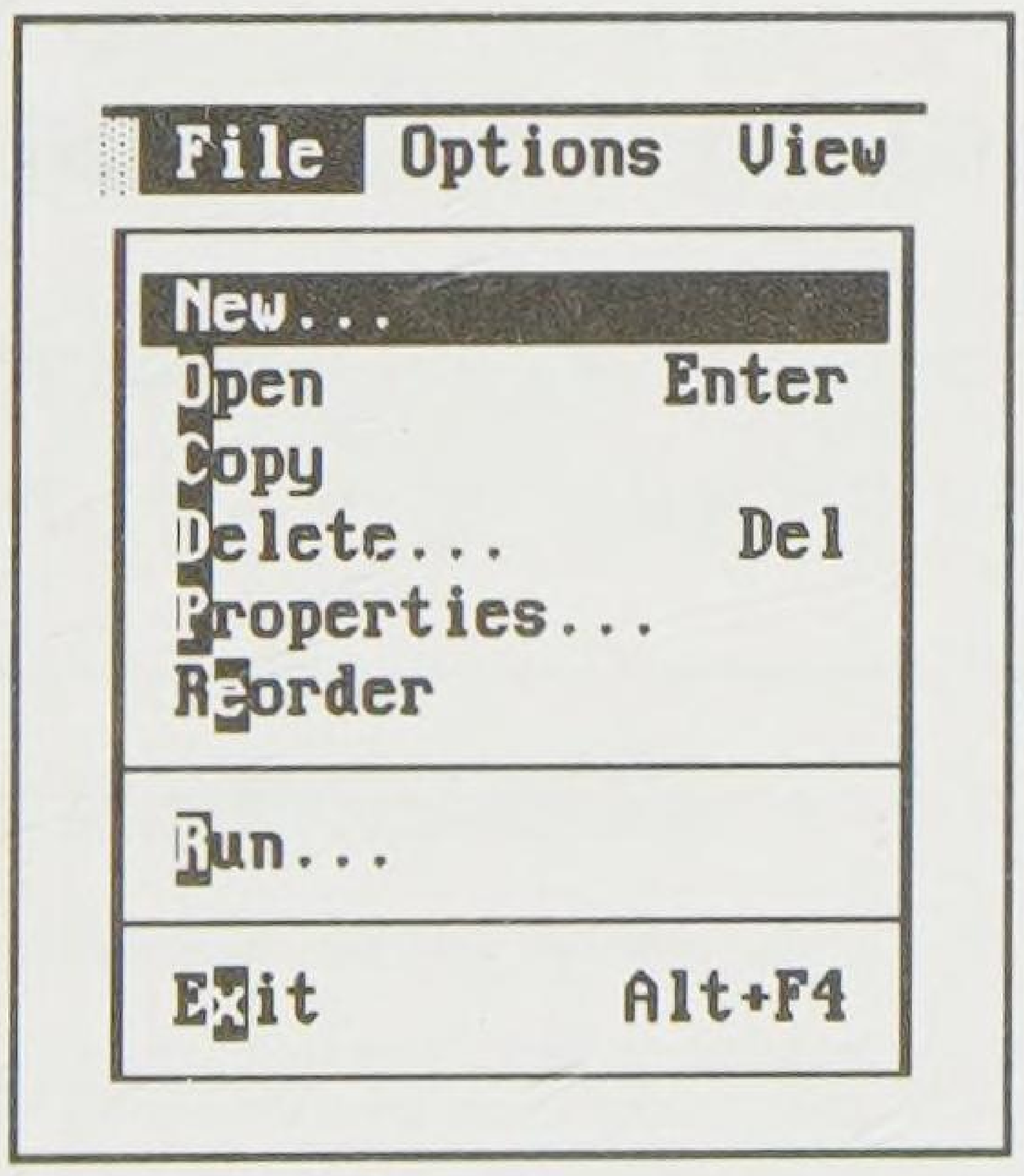
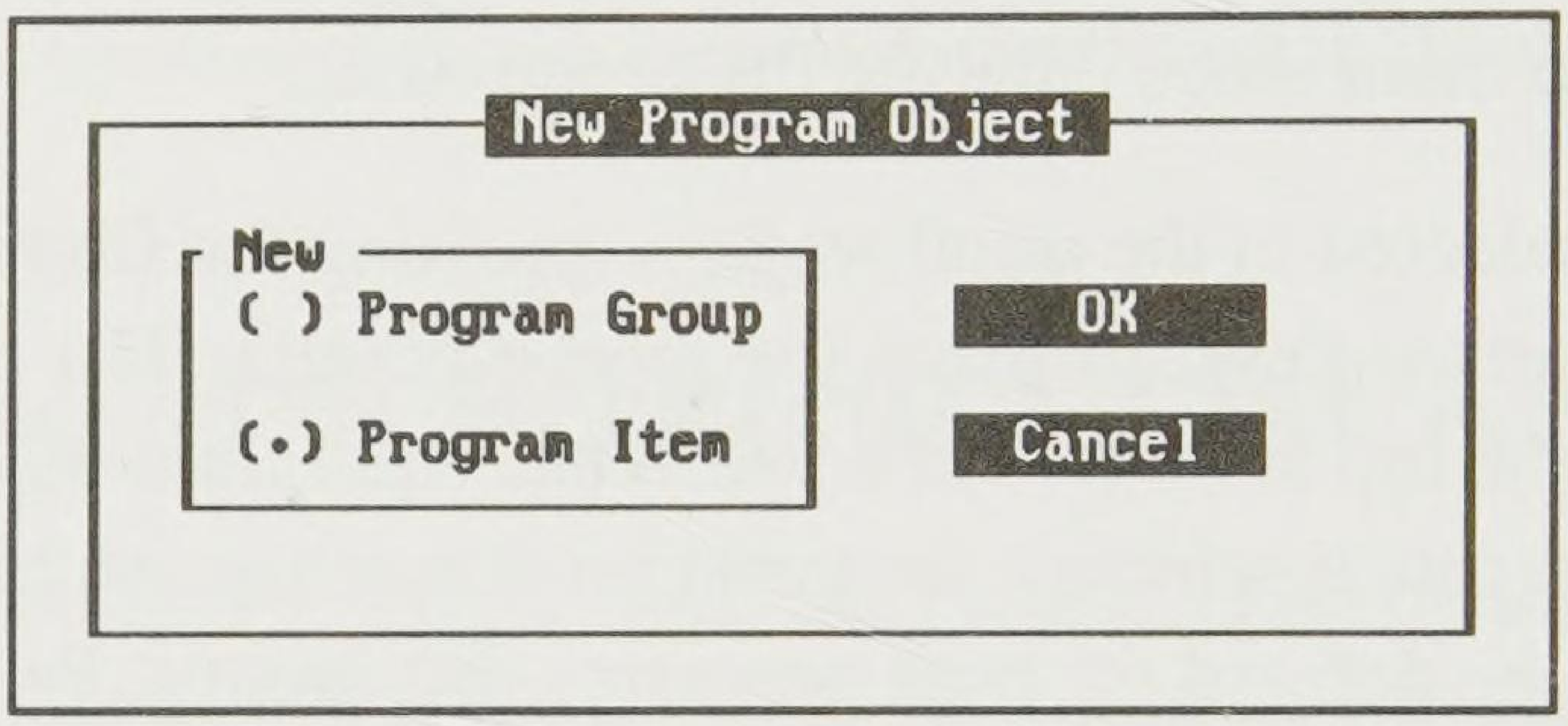


Figure 2-41 — File Menu when Program List Area is Active.

The Open, Run, and Exit options function in the same way as those on for the File List area. The other options are unique to the Program List area, and they are discussed in the following sections.

Programs and groups may be added to the currently displayed menu. When the Shell starts, the current Program menu is Main, as shown in Figure 2-1. To add items to a different menu, you must first display it. Programs and groups are both added by selecting New from the File menu. This brings up the dialog box shown in Figure 2-42.



( ) Program Group

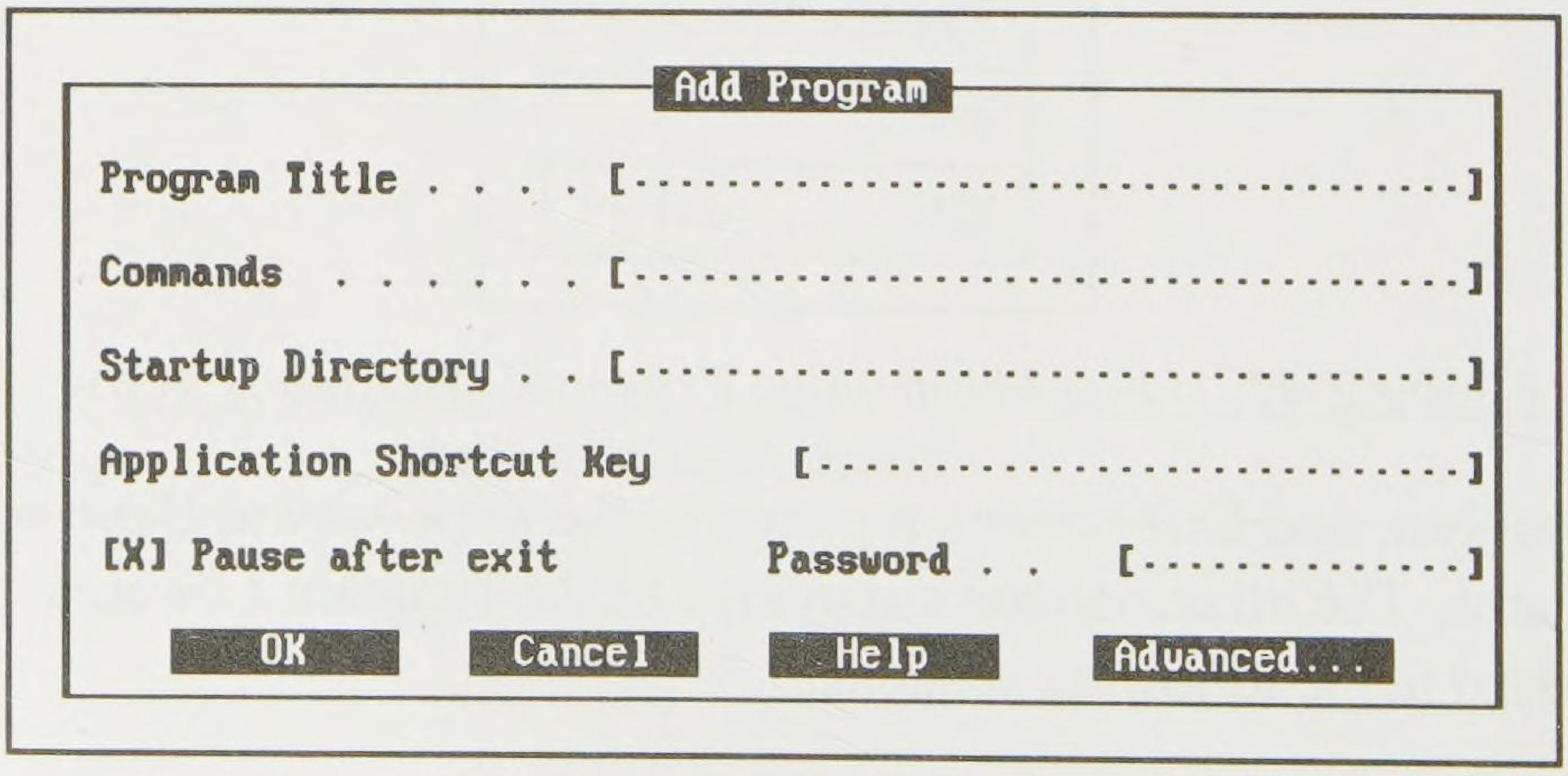
(•) Program Item

Figure 2-42 — Dialog Box for Specifying a Program or a Group.

When this dialog box appears, Program Item is the default selection. This will add a new program to the current menu. Selecting Program Group will add a new group title to the current menu. After selecting the type of object you want to add, press Enter← or select the OK button. You may also double click the mouse on the object type.

Adding a Program to a File List Menu

If you selected Program Item in the New Program Object dialog box (Figure 2-42), the Add Program dialog box shown in Figure 2-43 will be displayed.



Program Title . . . . [..................................]

Commands . . . . . . [..................................]

Startup Directory . . [..................................]

Application Shortcut Key [..........................]

[X] Pause after exit Password . . [.............]

Figure 2-43 — Dialog Box for Adding a Program to the Current Menu.

The Program Title is what will appear on the menu. It does not have to be the same as the actual program file name. In fact, it is usually better to use a longer and more informative title. The maximum length for a program title is 23 characters. For example, the DIR program that comes with DOS could be added to the Main menu. DIR is used to list the contents of a directory. An appropriate title might be, List a directory. After entering the title, press tab to move the cursor to the Commands text box.

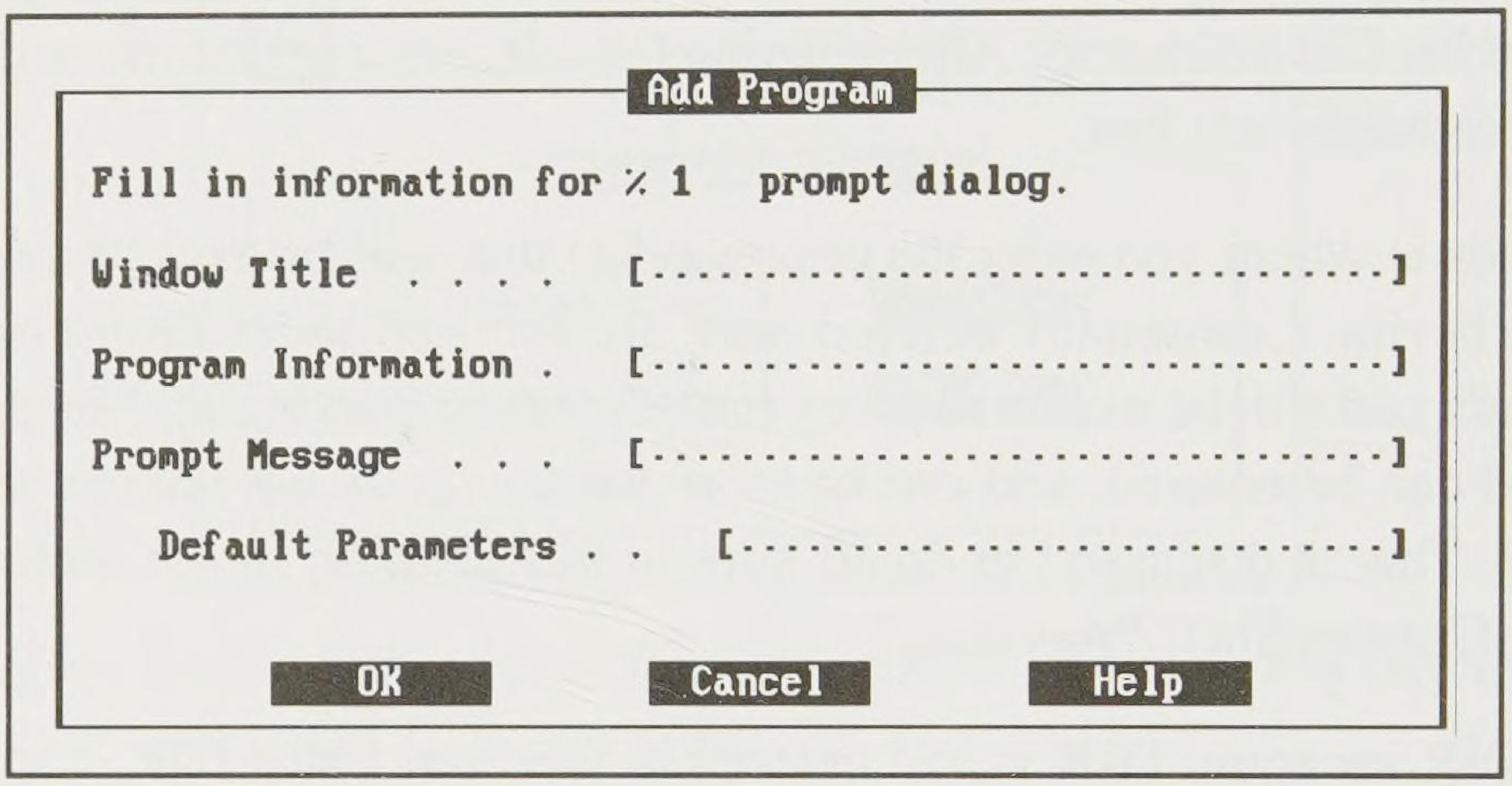
Commands is where you enter the command(s) that will be executed when this program is run. Commands entered here are referred to as Program Startup Commands and will be executed every time the menu item is run. More than one command can be entered, and one or more parameter dialog boxes can also be specified. This is discussed in detail later in this chapter, in the section titled, "Writing Custom Shell Programs."

For the DIR program, **DIR** would entered in this box. Since DIR is an internal DOS command, the Shell does not have to find it on the disk in order to run it. However, if the program specified is on the disk, as is usually the case, be sure the program file is either in the current directory when the program is run, or that it is in a directory included in the current path. Paths are discussed in Chapter 5.

Parameter dialog boxes are used to prompt the user for information that will be passed to the program. For example, when a word processor is started it is often desirable to pass the name of the file to be edited. This saves the steps involved in loading the file after the word processor screen appears. You may specify up to nine parameters, each with its own custom dialog box. To have a parameter dialog box pop up when the program is run, enter a percent sign followed by a number from 1 to 9 in the Commands text box. In the DIR example, it would be convenient to prompt the user for the name of the directory to list — otherwise the program would always list the current directory. To add a parameter dialog box to the DIR command, change the text in the Commands text box to:

```
DIR %1
```

Be sure to leave a space between the **R** and the % sign. If one or more parameters are specified in the Commands text box, then after this dialog box is accepted (by pressing Enter← or selecting the OK button), the Program Item Properties dialog box in Figure 2-44 will be displayed.



Fill in information for % 1 prompt dialog.

Window Title . . . . [································]

Program Information . [································]

Prompt Message . . . [································]

Default Parameters . . [····························]

Figure 2-44 — Dialog Box for Defining Program Parameters.

This dialog box allows you to specify how the parameter dialog box should look. The title that will appear at the top of the parameter dialog box is entered in the Window Title text box. A title of up to 23 characters may be entered here. An appropriate title for the DIR example would be, Directory To List. The Window Title is optional, but if you leave it blank the Shell will fill in a title that consists of the name of the command and the parameter specifier; for this example this would be DIR %1. This is not a very user-friendly title, so it is generally a good idea to provide your own.

The Program Information text box is used for an optional message that will be shown at the top of the dialog box. This is usually used to indicate precisely what the user is expected to enter. A maximum of 106 characters may be entered here. In the DIR example, a reasonable message would be; "Enter the name of the directory to list. Be sure to indicate the complete path."

The Prompt Message is the text that will appear to the left of the text box used for capturing the user's response. In Figure 2-44, "Window Title" is the Prompt Message for the first text box. For the DIR example, "Directory Name" would be an appropriate Prompt Message. The Prompt Message may be left blank.

The Default Parameters text box is used to specify a value that will be used if the user does not enter a parameter. This may be left blank, in which case no parameter is passed to the program unless the user enters one. In the DIR example, the root directory could be specified as the default parameter by entering a backslash here \.

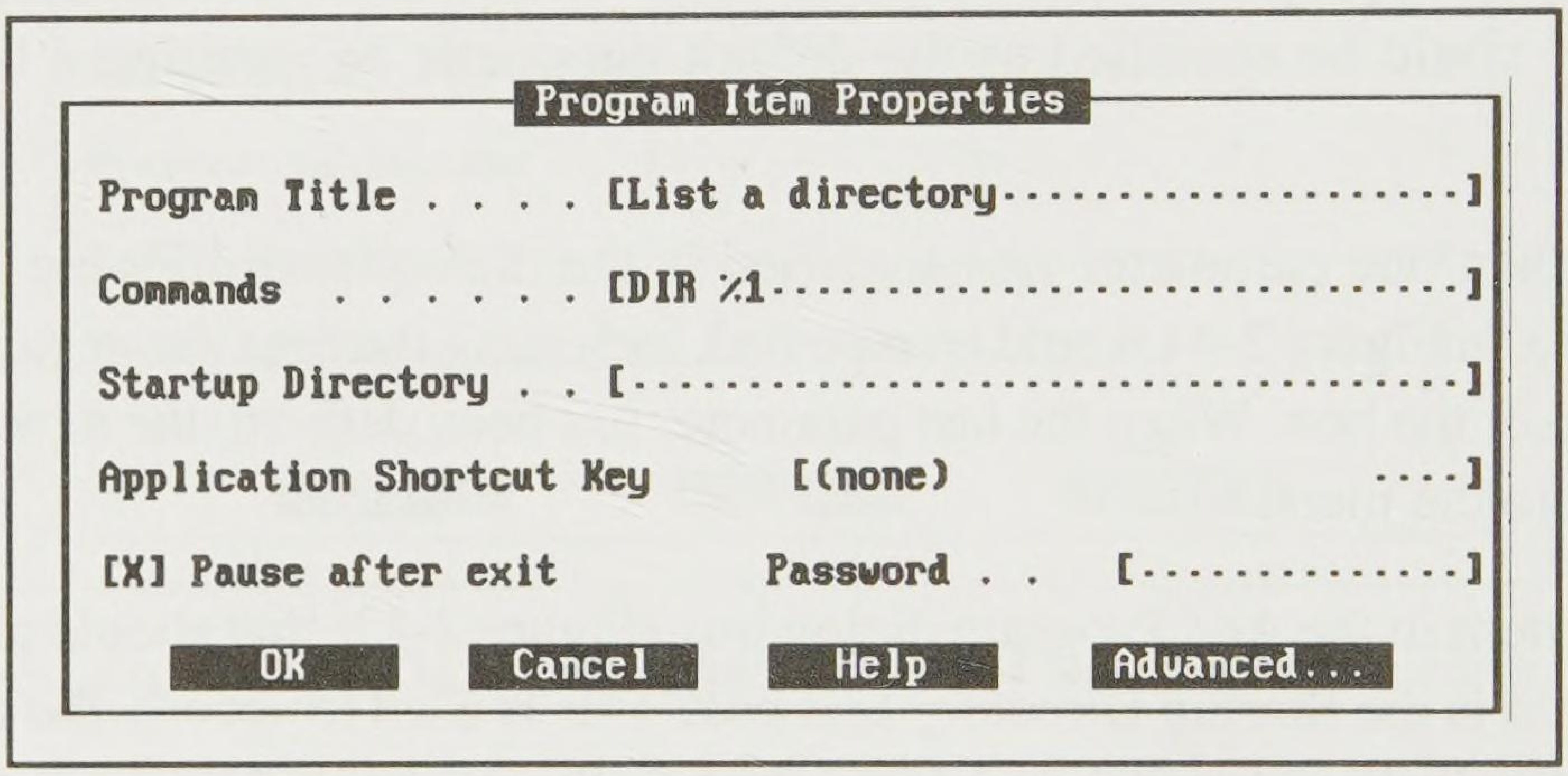
If more than one parameter was specified in the dialog box in Figure 2-43, the dialog box in Figure 2-44 would be repeated, indicating the next parameter number at the top of the box. When the last parameter has been defined, the new program is added to the menu.

The last item in the Add Program dialog box (Figure 2-43) that should always be considered is the Startup Directory text box. This is used to specify the directory that the Shell should switch to before starting the program. For example, if the program being added were a word processor in the C:\WP directory, and there was a subdirectory named DATA that contained the data files for the word processor, then the Startup Directory would be C:\WP\DATA. For the DIR program in the example above, this line should be left blank, since the user will specify a directory in the parameter dialog box.

There are several other options available in the Add Program dialog box. However, they can be added either when the program is defined, or later, by choosing the Properties option on the File menu. Each of these options is discussed in the following sections.

Modifying a Program in a Program List Menu

Selecting a program in the Program List area, then choosing the Properties option on the File menu, allows you to modify the way the program is defined. This brings up a dialog box that is almost identical to the Add Program box (Figure 2-43), except that the defined areas of the box are already filled in. This box, with the information for the DIR example, is shown in Figure 2-45.



```
Program Item Properties

Program Title . . . . [List a directory···················]
Commands  . . . . . . [DIR %1·····························]
Startup Directory . . [···································]
Application Shortcut Key     [(none)                  ····]
[X] Pause after exit        Password . .   [··············]
```

Figure 2-45 — Dialog Box for Modifying Program Properties.

Defining an Application Shortcut Key

The Task Swapper, discussed in detail later in this chapter, allows you to run multiple programs at the same time, and to switch between them. To make it easy to switch to a program, you may specify a "shortcut key." This shortcut key cannot be used to start the program, but once the program is started and running under the Task Swapper, the shortcut key can be used to quickly switch to it. The shortcut key is specified in the "Application Shortcut Key" text box. A shortcut key is a combination of the alt, Ctrl, or ⇧Shift keys, and almost any other key. For instance, holding down alt and pressing W would define alt-W as the shortcut key for an application. When the two keys have been pressed, the Shell displays their combination in the text box. For example, holding down alt and pressing W would cause **Alt+W** to appear in the text box. You may use any combinations of the alt, Ctrl, and ⇧Shift keys with any other key. For instance, holding down alt and Ctrl, together with home defines the hot key to be **Alt+Ctrl+Home**.

There are a few key combinations that cannot be used for application shortcut keys. If you press one of these keys, the Shell will either not respond at all, or it will cause the computer to beep.

To remove a shortcut key that has already been defined, either replace it, or press delete. The term "(none)" will appear in the Application Shortcut Key text box after the key has been deleted.

When a program has a shortcut key, and the program is running under the Task Swapper, the shortcut key will appear next to the program title in the Active Task List.

Pause After Exit Option

The default for the "Pause after exit" check box is on. This causes DOS to pause when a program is terminated, and to display the message "Press any key to return to MS-DOS Shell..." Turning this check box off will cause the Shell to return immediately after the program terminates. Many programs, particularly DOS commands, produce some kind of output that you will want to review before returning to the Shell — it is in these circumstances that it is useful to have this pause after exit.

Adding a Password to a Program

Programs can be protected by passwords. The ability to include a password for your program allows you to provide security on your system. If your program permits access to confidential data then a password is a good idea. It can be used to prevent anyone but yourself and other authorized individuals from viewing the data or using the program.

The password is defined in the Password text box shown in Figure 2-45. Passwords may be up to 20 characters long and may include any alphabetic or numeric character, as well as spaces. Passwords are case sensitive. When you type the password into the Password text box, it appears as you type it. However, when you enter the password to run the program the password will not be shown on the screen. This is to prevent other people from seeing it.

LoOk

Passwords are entirely optional and it is generally not a good idea to use them unless they are really necessary. They slow down the opening of a program, since the Shell must pause and request that you type in the password each time the program is run. There is also the risk that the password will be forgotten.

If a password is defined for a program, you will be asked to enter the password not only for running the program, but also to modify it. If you forget the password it will be impossible to get at the Add Program screen to put in a new password, so you must be sure to use passwords you will remember, or write them down somewhere they can be looked up.

When a program is password protected, whenever the program is accessed — either to run or to modify it — the Password dialog box, shown in Figure 2-46 is displayed.

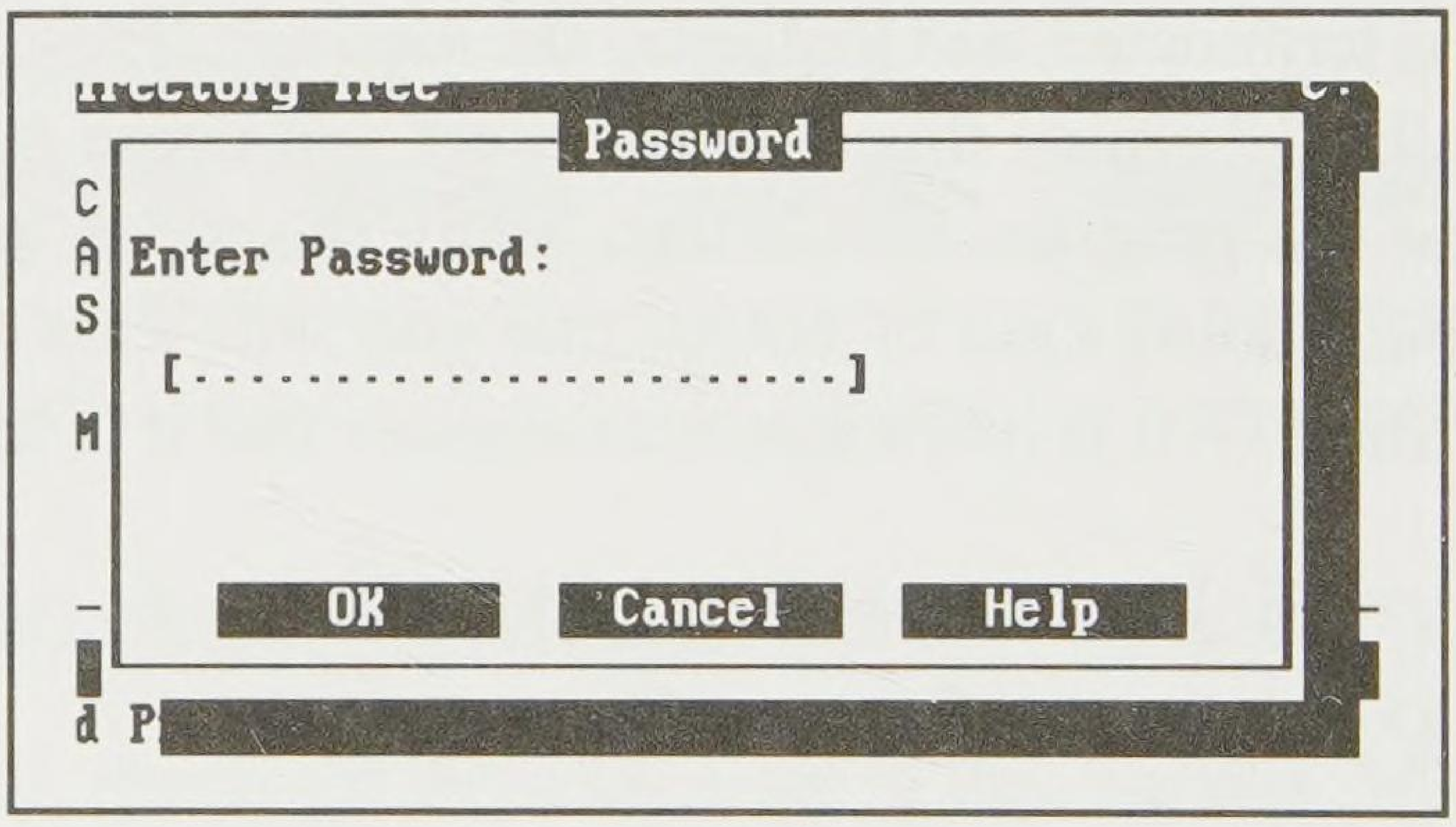


Figure 2-46 — Password Dialog Box.

When the password is typed in, it does not appear in the text box. Instead, a series of asterisks appear, one for each character typed. This allows you to see how many characters you have typed, but does not permit prying eyes to see your password. If the wrong password is entered, the computer beeps and you are returned to the Shell. If the correct password is entered, you are granted access to the program.

Recovery from Lost Passwords

The password capability is designed to provide protection against unauthorized use of a program, but if you forget your password it can leave you stranded, since you cannot get into the program definition to change the password. However, if you understand how the Shell stores passwords, there is a way to recover from this situation. Unfortunately, this also means that anyone who knows the method can use it to break into your programs. Consequently, the Shell password system should not be relied on when a high level of security is required. It is more suitable for keeping the casually curious user from accessing your programs.

The Shell remembers many things about your configuration: the color setup you have selected, the view you were in when you last left the Shell, the passwords for your programs and menus, etc. It does this by writing this information to a file named DOSSHELL.INI, in the DOS directory. DOSSHELL.INI will be examined in greater detail in the Advanced Topics section later in this chapter. For now what is important to know is that, for each program you add to the File List area, there is a program definition section added to DOSSHELL.INI.

For the purposes of this discussion, the password abcxyz was created for the DIR program. The program definition for this example would be similar to that shown in Figure 2-47.

```
program =
{
   command = DIR %1
   title = List a directory
   pause = enabled
   dialog = dialog
   password = GHIDEF
   dialog =
   {
      title = Directory To List
      info = Enter the name of the directory to list.  Be sure to include the
             complete path.
      prompt = Directory Name
      default = \
      parameter = %1
   }
   help = This is the help text for the DIR program.  ^mDIR is used to list the
          contents of a directory.
   screenmode = text
   alttab = enabled
   altesc = enabled
   ctrlesc = enabled
   prevent = disabled
}
```

Figure 2-47 — Section Of DOSSHELL.INI for DIR Program.

DOSSHELL.INI is a plain text file, which means it can be examined and modified by any text editor. You should NOT modify it unless you are absolutely certain of what you are doing. Even then, you should create a copy of it with the DOS copy command, or the COPY feature on the File menu, before making any changes. That way you can always restore the original from the copy if you make a mistake you cannot undo.

DOSSHELL.INI is a fairly large file, but you can quickly locate the section for a particular program by performing a text search on the phrase "command = xxx," where xxx represents the command entered in the Commands text box. In Figure 2-47, the line is "command = DIR %1."

The line you are looking for is the one reading: password = GHIDEF. Note that GHIDEF is not the password you entered for this program. The reason it is different is that the Shell encrypts it to protect against prying eyes. If you wanted to spend enough time and effort, you might be able to figure out how this encryption scheme works, it is relatively simply. But there is an easier way to get around the problem: delete the password. To eliminate password protection for the program, find the line containing the password (password = GHIDEF), and delete the line. Then save the file.

Note: When you save the file, make certain you save it as an ASCII file (or DOS text file). Otherwise your word processor may put special codes into the file that make it unusable by the Shell. Another potential problem with editing the DOSSHELL.INI file is that it may contain lines that have more than 256 characters on them. Some word processors will either split these lines or truncate them. You must be certain your word processor can manage lines longer than 256 characters before using it to edit this file. One way to avoid these problems is to use the MS-DOS Editor that comes with DOS. This always produces ASCII files (there is no danger that hidden codes will be added) and it can handle lines longer than 256 characters.

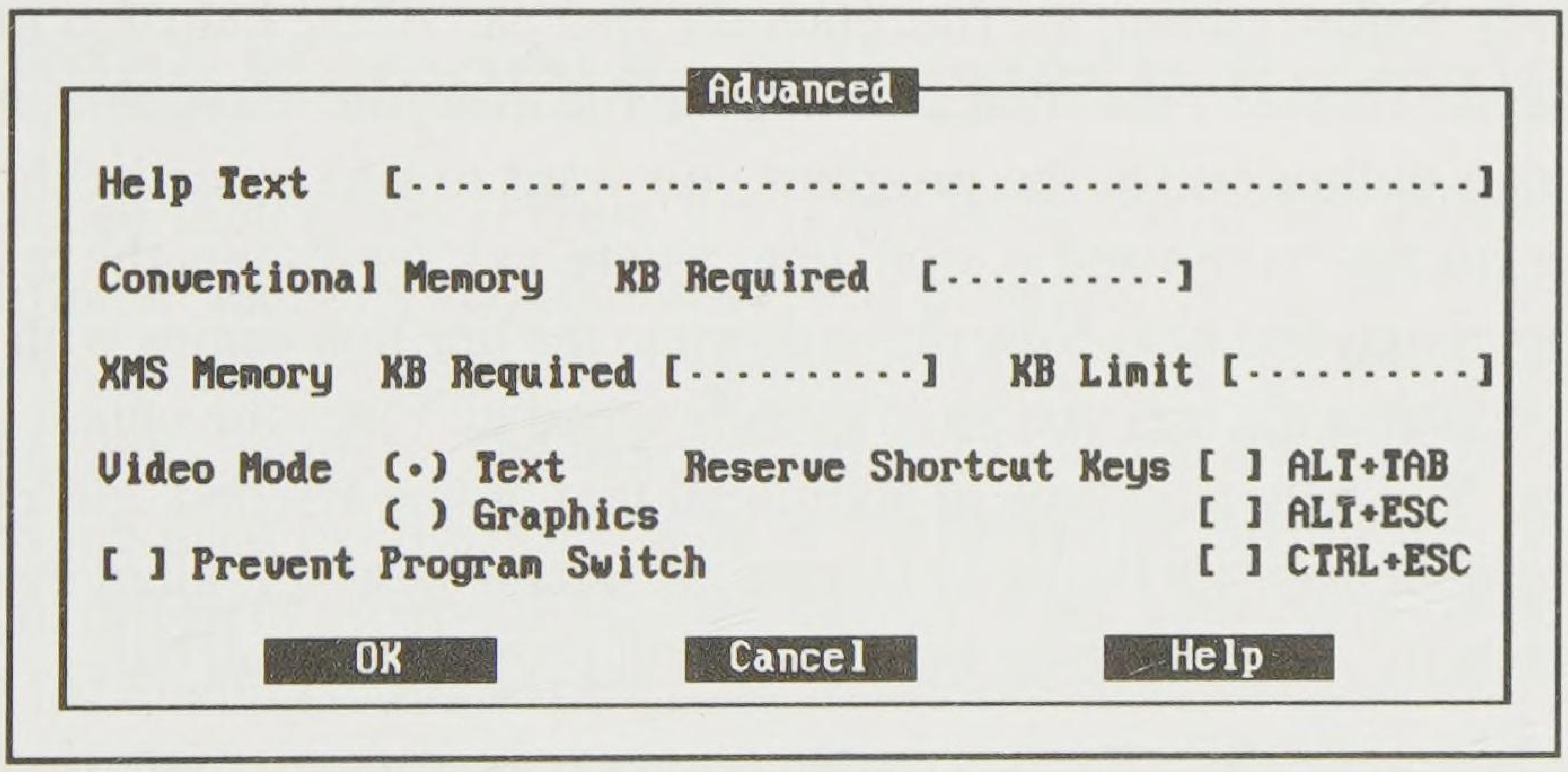
Warning: You must completely exit the Shell before editing DOSSHELL.INI. This is because DOSSHELL.INI is written to disk when you exit the Shell. If you edit the file while the Shell is running, the edited file will be overwritten when you exit the Shell. This will restore the password you just edited out.

Providing Online Help for Programs

You have already seen that online help is available for the Shell's various built-in programs and functions. One of the advantages of adding programs to the Program List is that you can provide similar help screens for them.

In Figure 2-45, there is an option button labeled "Advanced ..." at the bottom of the screen. The ellipsis (...) indicates that selecting this button will bring up another dialog box. Advanced options are features that can be used to fine-tune the way a program interacts with the Shell.

To add help text for a program at the same time you are adding the program to the Program List, select the "Advanced ..." button in the lower right corner of the "Add Program" dialog box to bring up the "Advanced" dialog box shown in Figure 2-48.



```
                              Advanced
Help Text   [.........................................]

Conventional Memory   KB Required  [..........]

XMS Memory  KB Required [..........]   KB Limit [..........]

Video Mode   (•) Text     Reserve Shortcut Keys [ ] ALT+TAB
             ( ) Graphics                       [ ] ALT+ESC
[ ] Prevent Program Switch                      [ ] CTRL+ESC

        OK                 Cancel                Help
```

Figure 2-48 — Advanced Options Dialog Box for Shell Programs.

To add or modify help text for a program that is already in the Program List, use the following steps:

1. Select the program.
2. Press alt-F to pull down the File menu
3. Press P to bring up the Program Item Properties dialog box.
4. Select the "Advanced..." button in the lower right corner. This displays the dialog box shown in Figure 2-48.

Help text for the program can be entered in the "Help Text" text box. The text may be up to 256 characters long. Help text is often more readable if it is separated into two or more paragraphs. Since the text entered in the Advanced dialog box is one long string, a special code is required to force the text to drop down a line. The special code is the combination of the caret, ⇧Shift-6 on the numeric keys across the top of the keyboard, followed by a lowercase m: ^m. The caret immediately preceding the m represents the combination Ctrl-M, which is an ASCII carriage return (0Dh); but that is not how this symbol is produced — you must type the caret key and the **m** key separately.

There are two ways of getting around the limit of 256 characters: the first is to use a text editor, such as the EDITOR program that comes with DOS 5.0, to edit the DOSSHELL.INI file. Editing DOSSHELL.INI is discussed earlier in this chapter, in the section on "Recovery From Lost Passwords." (Be sure to review that section before editing DOSSHELL.INI; there are potential dangers that must be avoided). Before editing the file, enter the first part of the help text in the Help Text text box. Then exit the shell and load the file into your text editor, and locate the program definition for the program you want to add extended help to, by searching for the "command = xxx" line, where xxx represents the text entered in the Commands text box. Now move down to the line that begins with "help =". This line contains the text you have already typed in. Place the cursor at the end of the line. You may now type in additional text, going beyond 256 characters. The ^m symbol may still be used to create paragraph breaks. If all of the text does not fit inside the box, the Help dialog box will provide a scroll bar.

You must be certain to entirely exit the Shell before editing DOSSHELL.INI. Otherwise, when you exit the Shell after editing DOSSHELL.INI, your changes will be lost.

There is one danger with this technique: if you have extended the Help text to more than 256 characters, then bring up the Program Item Properties screen for the program, then go to the Advanced screen, you will only see the first 256 characters. If you then return to the Program Item Properties screen and select OK, or press Enter↵, only the first 256 characters of text will be saved. If you don't go to the Advanced screen, or if you select Cancel instead of OK on the

Program Item Properties screen, then you will not affect the help text. This means that, once you have defined help text of more than 256 characters, you cannot use the Advanced screen to change any of its options.

Because of this limitation, a better approach is to provide multiple, linked screens. On some of the help screens in the Shell, you may have noticed a selectable item labeled "Next topic," "Related procedure," or some similar phrase, as in Figure 2-6. When these items are selected, another help dialog box is shown with information about the selected topic. The "Advanced Topics" section, later in this chapter, discusses techniques for providing "Next topic" connections in your own help dialog boxes.

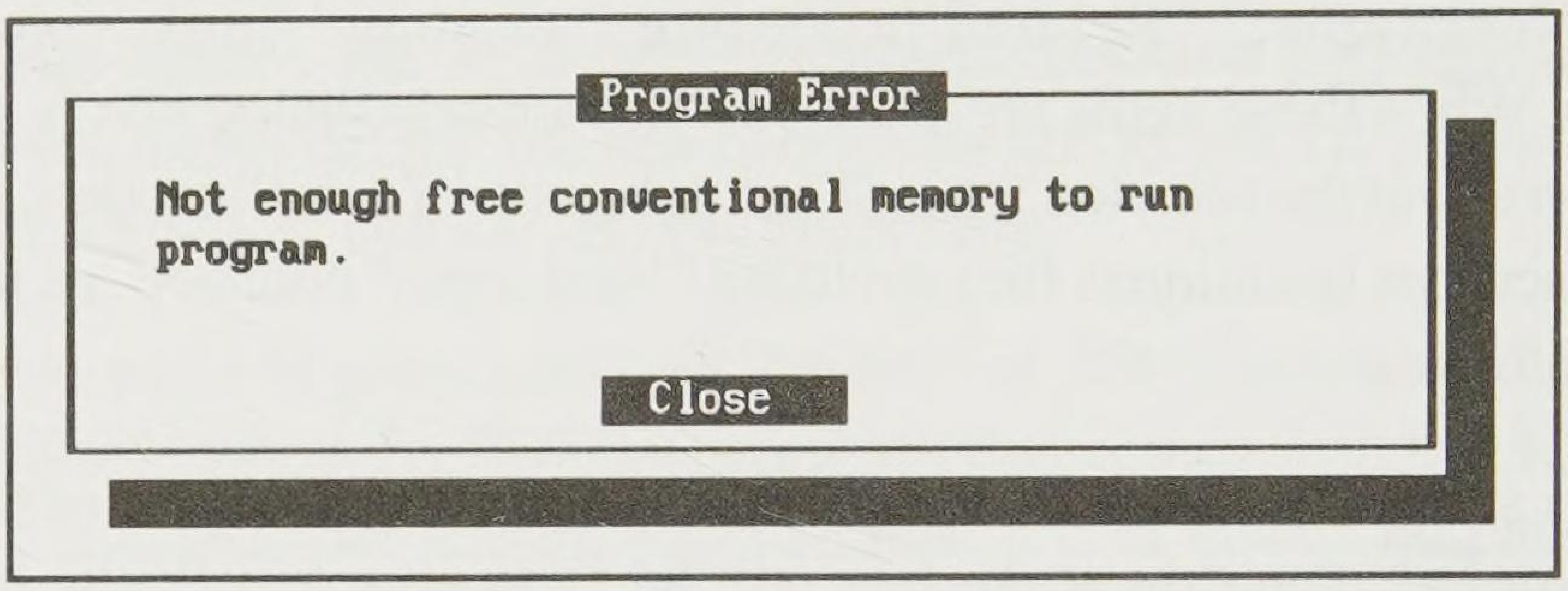
Help text is accessed from the Program List by selecting the name of the program and pressing F1. The title of the box will be "Help For xxx," where xxx is the title of the program.

Specifying Memory Requirements for Programs

The Advanced dialog box (Figure 2-48) has three text boxes where you may specify different memory requirements for a program. When the Task Swapper is used to run multiple programs concurrently, each program must be given a piece of the computer's memory. The memory specification text boxes in the Advanced dialog box are used by the Task Swapper for balancing the amount of memory provided to different programs that are running at the same time. Since they are only used for running multiple programs simultaneously, these specifications are ignored when the Task Swapper is not enabled. A detailed discussion of computer memory is provided in Chapter 4. The Task Swapper is discussed later in this chapter, in the section titled, "The Task Swapper."

The first memory specification text box, "Conventional Memory KB Required," is used to specify the minimum amount of memory required to run the program. The number typed in here will be interpreted as Kilobytes (KB). For example, typing in 256 would mean that at least 256 Kilobytes of memory must be available to run this program under the Task Swapper.

The minimum specified for running the program does not affect the actual amount of memory the Shell will provide — the Shell always gives programs all of the available memory. In other words, if 256K is the minimum, but 400K is available, then 400K will be provided. If the minimum is not available, the Shell will not load the program, but instead will display the dialog box shown in Figure 2-49.



Not enough free conventional memory to run program.

Figure 2-49 — Dialog Box for Too Little Conventional Memory.

The minimum amount of memory required by a program is usually stated in the manual that comes with the program. If no minimum is specified, the Shell assumes a value of 128 KB.

The "XMS Memory KB Required" text box is for programs that require extended memory in order to run. Most programs do not require extended memory, so this box should be left blank if the program does not require extended memory. Putting a value here will significantly slow down the Task Swapper when it switches between programs. If no value is entered here, the Task Swapper uses 0 as a default. If the Task Swapper cannot find as much extended memory as is specified, a dialog box similar to that shown in Figure 2-49 will appear.

The "XMS Memory KB Limit" text box must be filled in if a program is to make use of extended memory. This text box specifies the maximum amount of extended memory the Shell will provide to the program. This is needed because many programs that use extended memory will take all that is available when they load, leaving no room for other applications. If this box is left blank, the program will be given 0 KB extended memory. Specifying minus one (–1) will give the program all that it asks for, up to the amount available. Specifying anything else will give the program what it asks for up to the specified amount, if that much is available. You should only specify –1 if the program requires a lot of extended memory.

Setting the Video Mode for a Program

The "Video mode" text box in Figure 2-48 refers to how much memory the Shell uses for video mode. It does not affect the actual display. In most circumstances this should be set to text (the default). However, if you have a CGA monitor and have trouble switching to a program, changing this to Graphics mode may help. EGA, VGA, and monochrome monitors should always be left in Text mode.

Disabling the Shell's Shortcut Keys

There is a set of three check boxes at the lower right side of the Advanced dialog box (Figure 2-48). The title for this set is "Reserve shortcut keys." The three check boxes correspond to the key combinations alt-tab, alt-esc, and Ctrl-esc. Each of these key combinations is used by the Task Swapper as a shortcut to one of its functions. Consequently, when the Task Swapper is turned on, these key combinations are not available to programs. By checking the boxes for these key combinations, their use in the Task Swapper is disabled, making them available to the program.

Preventing the Task Swapper from Switching a Program

Some programs will not behave well when the Task Swapper switches to or from them — this may cause the machine to lock up (forcing you to reboot) or the screen display may become scrambled. If this happens, you may lock the programs out of task swapping. This means that when you start the program, you cannot leave it to return to the Shell or to another application without terminating the program. The Task Swapper is disabled by checking the "Prevent Program Switch" box in the Advanced dialog box (see Figure 2-48).

Adding a Group to a Program List Menu

Besides adding new programs to a Program List menu, new groups may be added. A group is a set of items that appear on their own menu in the Program List area. An item in a group can be either a program, or another group. Groups are useful for providing a logical organization to the programs in the Program List. Programs that are related can all be grouped together, and if there is a very large group, you can build groups within groups, much in the same way a hard disk is organized into directories and subdirectories.

When you add a group, just the group title appears on the current menu. Selecting the group then brings up the menu of programs in the group. Groups can be easily distinguished from programs in a Program List menu because their names are surrounded by brackets. As seen in Figure 2-40, the Shell has one group pre-installed, named Disk Utilities. This is a good example of how to gather related programs into a group — all of the programs in this group are used for managing disks.

To add a new group, first make sure the Program List is the active area. Then select the "New..." option on the File menu. This brings up the New Program Object dialog box (Figure 2-42). Now select the Program Group button. The Add Group dialog box shown in Figure 2-50 will be displayed.

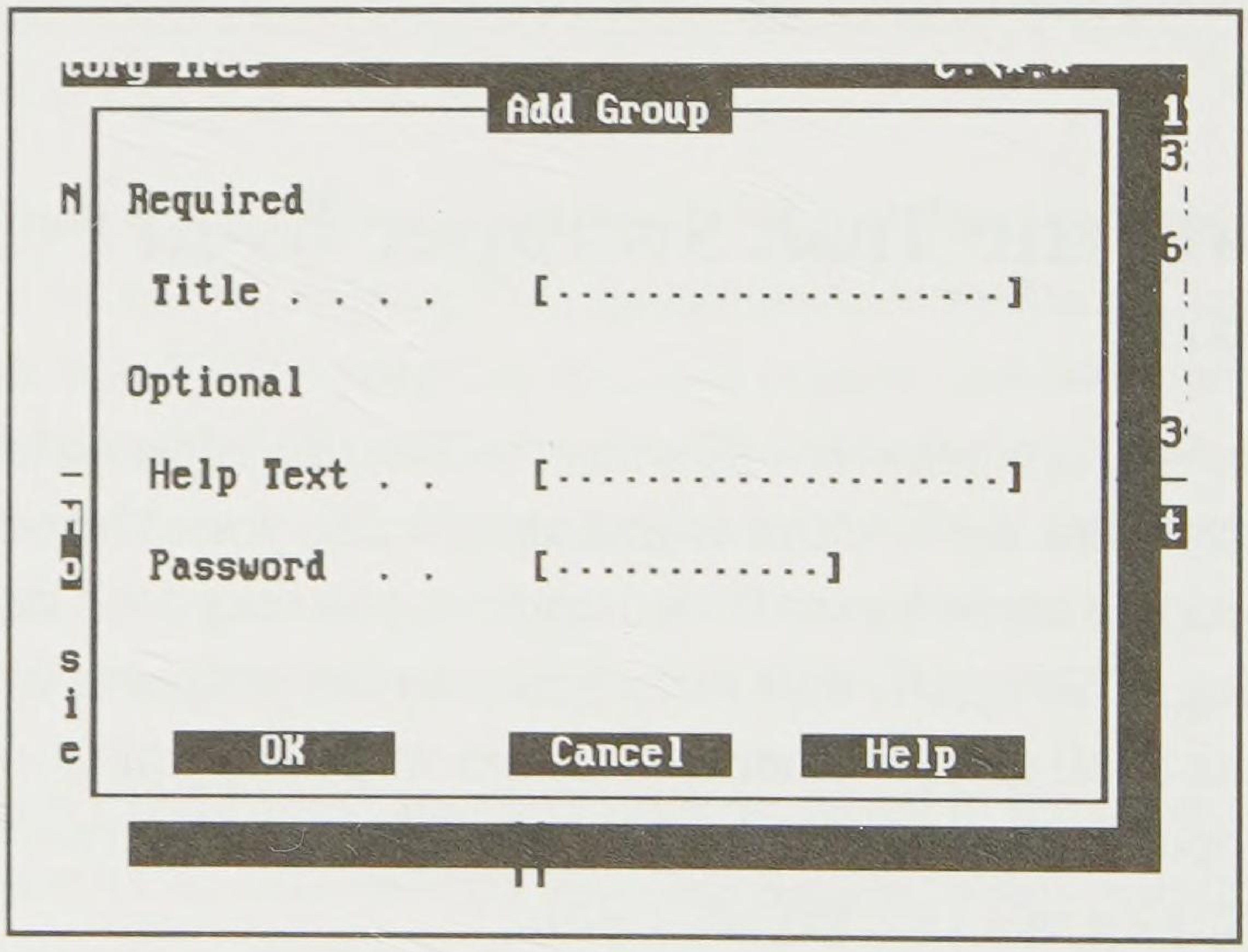

Add Group
Required
Title . . . . [....................]
Optional
Help Text . . [....................]
Password . . [............]
OK
Cancel
Help

Figure 2-50 — Dialog Box for Adding a Group to a Program List Menu.

A title for the group is required. This is entered in the Title box shown in Figure 2-50. This title is the name that appears on the current menu. The Help Text box is optional. Entering help text here is identical to entering help text for a program.

Passwords at the Group level restrict access to all programs within the group. This eliminates the need for setting up a password for each program. It also allows a hierarchy of passwords. For example, you can restrict access to Word Processing Functions to a particular set of users, then selectively restrict access within that set of users by providing additional passwords to the programs within the group. Entering passwords is discussed earlier in this chapter. This text box is optional and you should leave it blank if you do not want a password for the group.

When you add a new group to a menu, the group has one item in it — its parent group. To add more items to a group, you must first make it the current group on the screen. This is done by selecting it and pressing Enter←, or by double-clicking on it with the mouse. To demonstrate this, a group named "Word Processing" was added to the Shell File List. Selecting this group and pressing Enter← displays the screen shown in Figure 2-51.

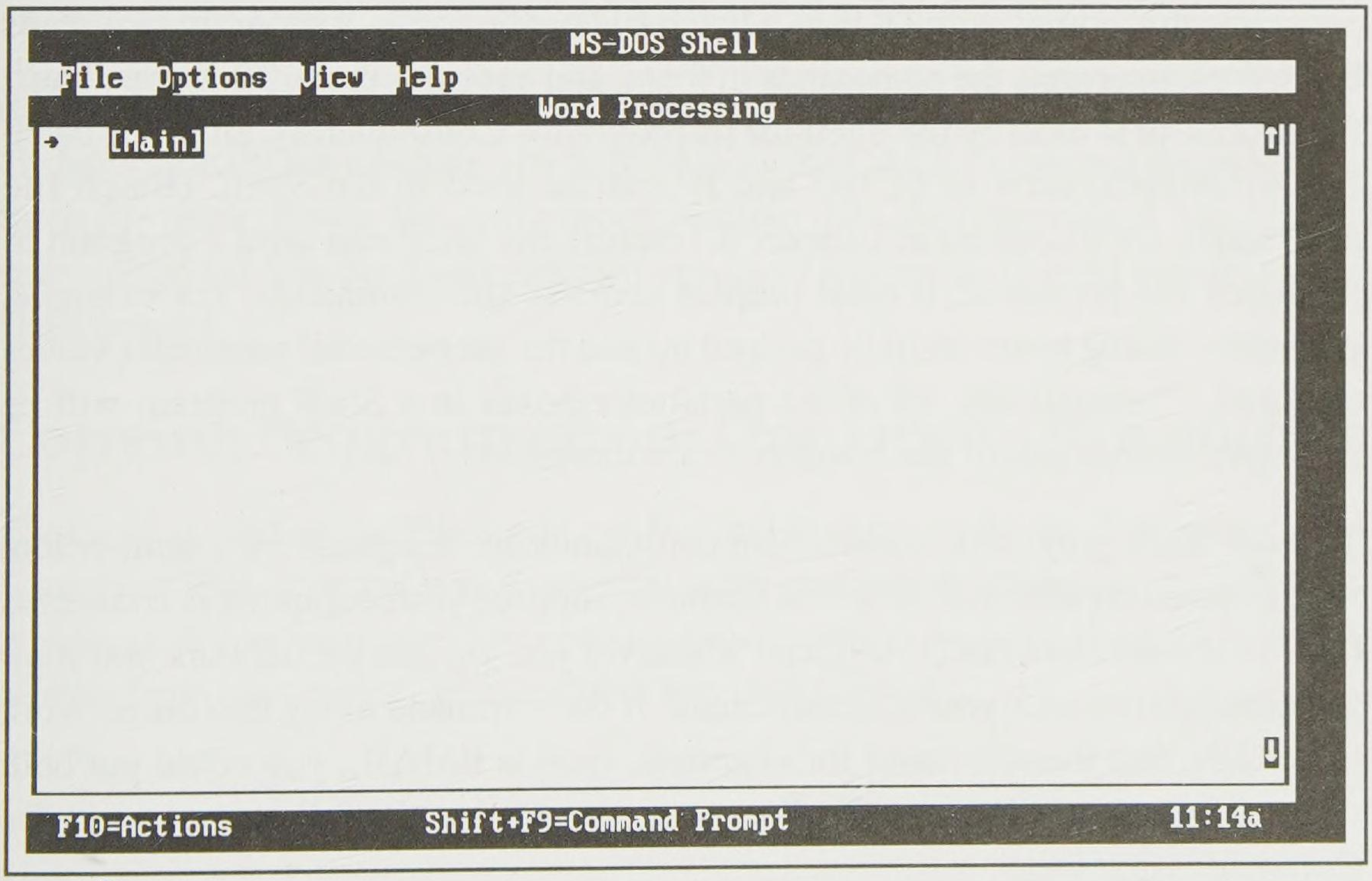


Figure 2-51 — Word Processing Group.

To add programs to a group, follow the directions listed in the "Adding A Program To A File List Menu" section earlier in this chapter; to add a group, follow the directions in this section.

Group titles, help text, and passwords can be changed by selecting the group title, then choosing the Properties option on the File menu. The "Program Group Properties" dialog box will be shown. This dialog box is identical to the "Add Group" dialog box (Figure 2-50), except for the title.

Writing Custom Shell Programs

In the examples discussed so far, a single command was entered in the Commands text box of the Add Program dialog box (Figure 2-43). However, it is possible to add multiple commands that execute in sequence, much like a batch file, thus creating Shell programs. Fairly powerful Shell programs can be written using DOS commands.

In order to understand how Shell programs work, it is helpful to know how the Shell executes commands that appear in the Add Program dialog box. One of the programs that comes with DOS is a batch file processor — a program that reads batch files, interprets the commands in them, and executes them. This same batch file processor is used by the Shell for its programs. Consequently, all of the batch file commands, such as ECHO and IF, can be used in the Shell. (Batch file commands are discussed in Chapter 8.) Before the Shell can send a program to the batch file processor, it must prepare some of the commands. For example, parameter dialog boxes must be popped up and the user-entered parameter values captured. Consequently, all of the parameter boxes in a Shell program will be displayed before any of the commands are executed.

To write Shell programs, consecutive commands are separated by a semi-colon, with a space on either side of it. For example, suppose your computer is connected to a Local Area Network (LAN), and whenever you log into the network you want to immediately check your electronic mail. If the command to log into the network is LOGIN, and the command for electronic mail is EMAIL, you could put both of these commands into one Shell program by putting the following text in the Commands text box:

```
LOGIN ; EMAIL
```

Batch programs can be executed from Shell programs by using the CALL command. For example, if a batch program is named WP, then the command `CALL WP` would cause the WP program to execute. After the batch program executes, control is returned to the Shell program, where any subsequent commands are executed.

To continue the above example, imagine that you know you will either want to run your spreadsheet or your word processor whenever you leave electronic mail, and that the commands for starting the spreadsheet are in a batch file named SS, and the commands for the word processor in a batch file named WP. A Shell parameter dialog box can be used to capture the program you want to run next. The following command line expands the one above to include this feature (in the Shell, this would be all on one long line):

```
LOGIN ; EMAIL ; IF %1 == WP CALL WP ; IF %1 == wp CALL WP ;
IF %1 == SS CALL SS ; IF %1 == ss CALL SS
```

If you include a batch file in a Shell program, and find that commands following the batch file do not execute, check to be sure the CALL command precedes the batch file name. Batch files will execute without the CALL command, but control does not go back to the Shell program you have written — control returns to the Shell instead.

Copying Programs from One Group to Another

The Copy option on the File menu can be used to copy a program from one group to another. This is useful if you want the same program to appear in two different groups, or if you want to move a program — in which case the program must be deleted from the original group after it has been copied.

The following steps should be followed to copy a program from one group to another:

1. Display the group that contains the program to be copied.
2. Select the program.
3. Press alt-F to pull down the File menu.
4. Press C to select Copy. The message "Display Group to Copy To, then press F2. ESC to cancel" will appear at the bottom of the screen.
5. Select and display the group that the program will be copied to. Note that it is not sufficient to highlight the group title in its parent group — you must actually display the group contents.
6. Press F2. The program you selected in step (2) will appear in the group list. Remember that the program still exists in the original group. If you only want it in the new group, you must return to the original group and delete it.

Changing the Order of Items within a Group

When items are added to a group, they are always added to the bottom of the list. This often leads to an arrangement of items that is less than ideal. It is particularly useful to put the most-used item at the top of the menu, because this is the item that is selected when the menu comes up. You can therefore run this program by pressing Enter← as soon as the group appears, without having to scroll through the list. The Reorder option on the File menu is used to rearrange the items in a Group list.

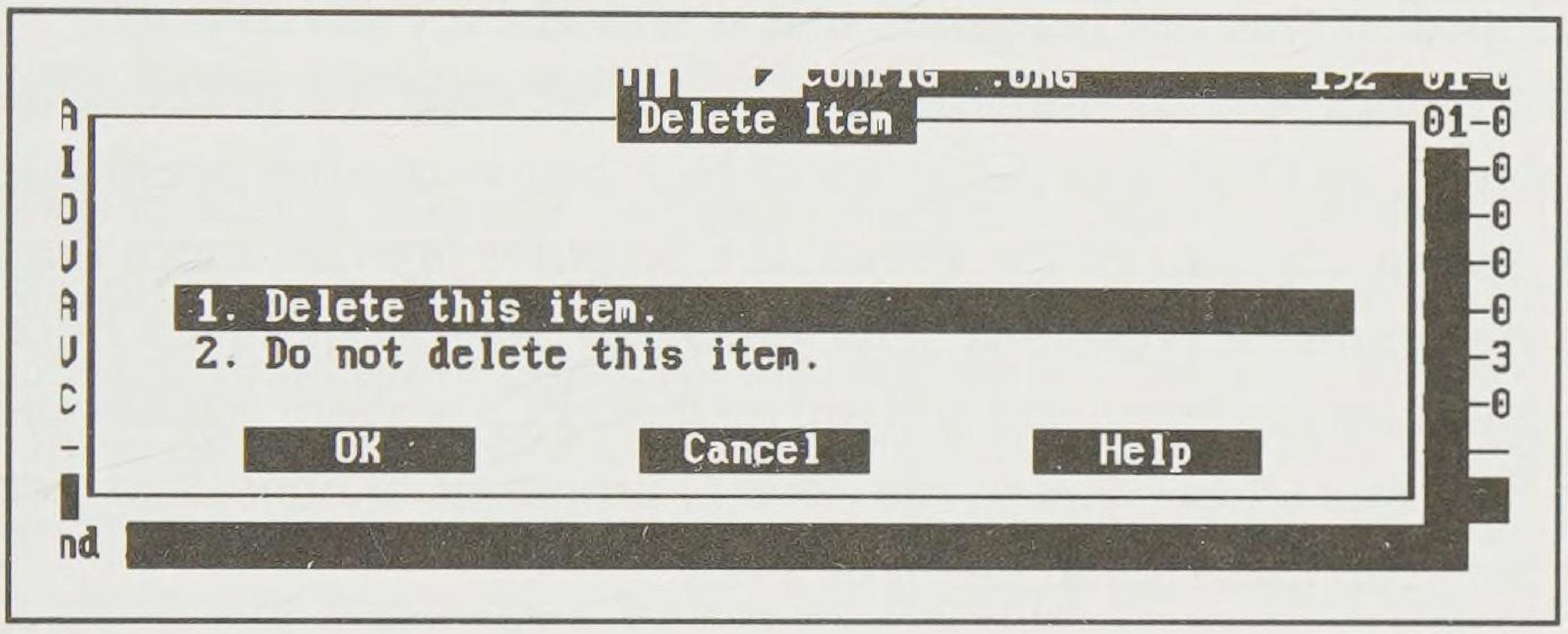
To move an item, first select it, then choose the Reorder option on the File menu. The following message will appear at the bottom of the screen:

```
Select location to move to, then press Enter. ESC to cancel
```

Now select the item that is in the position you want to move to, and press Enter←. The item will be moved to that position.

Deleting Programs and Groups

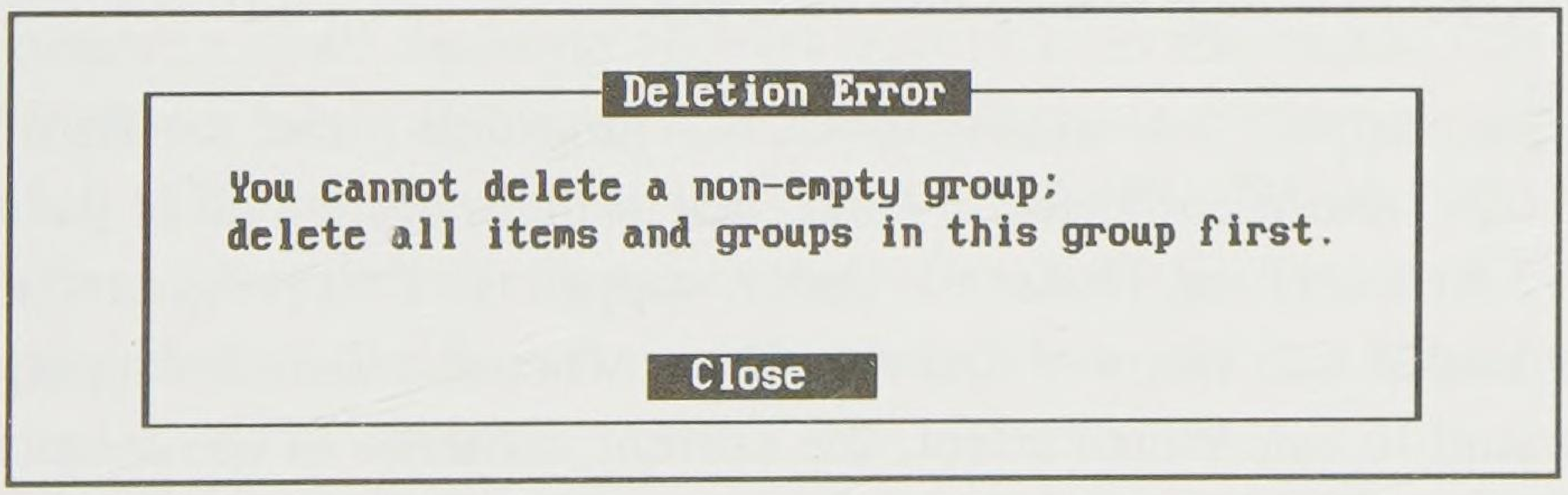
A program item can be deleted from a group simply by selecting the item, then choosing the Delete option (or pressing delete) from the File menu. The Delete Item dialog box shown in Figure 2-52 is displayed.



1. Delete this item.
2. Do not delete this item.

Figure 2-52 — Dialog Box for Deleting Programs from Groups.

Selecting option 1, then the OK button, deletes the program. Deleting a group follows the same process — select the group, then choose the Delete option on the File menu. However, the group must be empty before it can be deleted. This is a safeguard, to make sure you do not accidentally delete a group that has valuable programs in it. If you attempt to delete a non-empty group, the dialog box shown in Figure 2-53 will be displayed.



You cannot delete a non-empty group;
delete all items and groups in this group first.

Figure 2-53 — Dialog Box when Attempting to Delete a Non-empty Group.

If the group is empty, you will see the dialog box shown in Figure 2-52, to confirm that you want to delete the group.

The Task Swapper – Running More than One Program at Once

One of the most significant limitations of all previous versions of DOS has been their inability to run more than one program at one time. When the IBM PC was first developed, it was not imagined that it would ever develop into a machine powerful enough to run multiple programs at once (a process known as Multi-Tasking), so DOS was designed to be a single-tasking operating system. However, recent versions of the Personal Computer provide more than enough power to run multiple programs. The DOS 5.0 Shell provides a built-in Task Swapper that is easy to learn, and will run on most PCs without additional memory. The Task Swapper allows you to run several programs at once, and switch back and forth between them at the touch of a key

Since a single program can occupy most or all of the available memory in the computer, the Task Swapper is faced with the problem of providing that memory to whichever program you want to have active, while still remembering the status of the other programs that are open. It solves this problem by using the hard disk as "virtual memory." As you switch between programs you will notice that the computer will pause, and the light flash on your disk drive. What the Task Swapper is doing is taking a snapshot of the computer's memory, with the current program active, and writing a copy of the memory to a temporary disk file. The program being switched to is then loaded into memory.

For example, suppose you wanted to run two programs under the Task Swapper — Lotus 1-2-3 and WordPerfect — and each of them needed all of the available memory in the computer. Under the Task Swapper, the first program (say Lotus) would be loaded into memory and run. Then, when the Task Swapper received the command to run WordPerfect, the current contents of the memory (with Lotus 1-2-3 loaded) would be written to a temporary disk file before WordPerfect was loaded into memory. Now WordPerfect is running. When you ask the Task Swapper to switch back to Lotus 1-2-3, the contents of memory with WordPerfect loaded are saved in a disk file, and the disk file that contains the memory contents for Lotus 1-2-3 is read back into memory. In this way the Task Swapper can permit many programs to share the memory.

This approach to Task Swapping is effective and universal. It will work on any IBM-compatible Personal Computer with a reasonable amount of memory, since the Shell and the Task Swapper themselves do not require too much memory to manage the programs. However, since this approach requires intensive use of the hard disk to switch between programs, switching is slow. It also does not permit the transfer of data between applications — you cannot cut information out of one program that is running under the Task Swapper and paste it into another.

For contexts in which frequent switching between applications is not necessary, and transferring data between applications is not needed, the Task Swapper is an effective tool. If these more powerful capabilities are needed, more sophisticated software tools like Windows or DESQview are available. These programs allow many applications to be stored in memory at the same time, by making use of the larger memory capacities found on modern PCs. This permits almost instantaneous switching between programs, as well as the ability to cut and paste information from one program to another. However, there is a price to be paid — to work effectively these programs require significantly more powerful PCs, with more memory, than the Task Swapper requires.

Turning the Task Swapper On and Off

The Options menu, on the Shell menu bar, has an option titled, Enable Task Swapper. When you select this option and press Enter←, or click on it with the mouse pointer, a small diamond appears next to it on the menu. This indicates the Task Swapper has been turned on. Selecting it again and pressing Enter←, or clicking on it again with the mouse, will turn it off, and the diamond will disappear. (You will have to reselect the Options menu to see that the diamond has appeared or disappeared, because the menu disappears when you turn the Enable Task Swapper option on or off.)

If the current view is set to either Program/File Lists or Program List, both of which include the Program List area, then when you enable the Task Swapper, the screen will change to include the Active Task List. If the current view is Program File Lists, then the screen will appear as shown in Figure 2-54, with the Active Task List in the lower right corner.

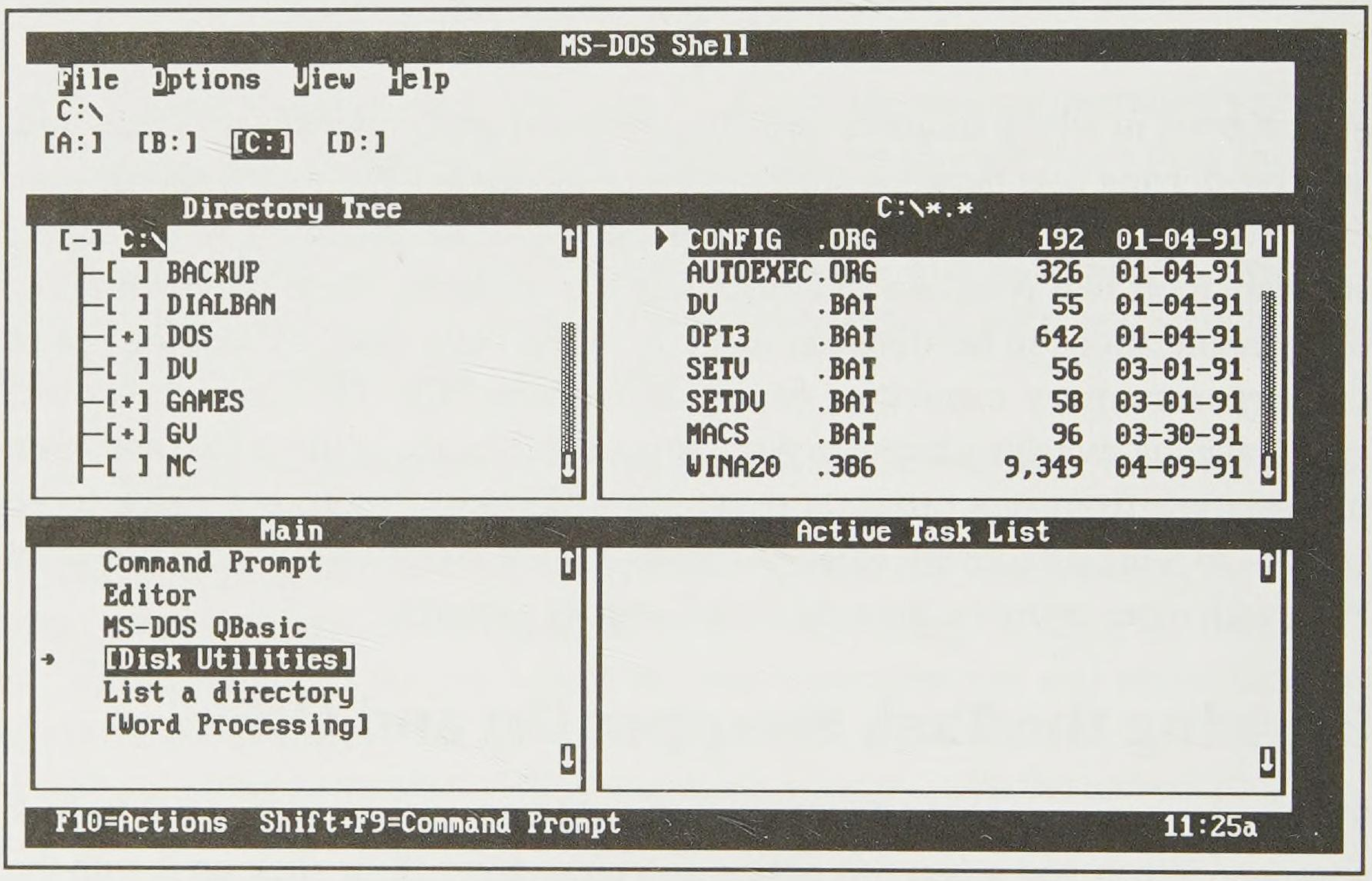

```
                              MS-DOS Shell
 File  Options  View  Help
 C:\
[A:]  [B:]  [C:]  [D:]
         Directory Tree                              C:\*.*
 [-] C:\                            ▶ CONFIG   .ORG        192  01-04-91
   ├[ ] BACKUP                        AUTOEXEC.ORG        326  01-04-91
   ├[ ] DIALBAN                       DV      .BAT         55  01-04-91
   ├[+] DOS                           OPT3    .BAT        642  01-04-91
   ├[ ] DV                            SETV    .BAT         56  03-01-91
   ├[+] GAMES                         SETDV   .BAT         58  03-01-91
   ├[+] GV                            MACS    .BAT         96  03-30-91
   └[ ] NC                            WINA20  .386      9,349  04-09-91
               Main                            Active Task List
     Command Prompt
     Editor
     MS-DOS QBasic
  →  [Disk Utilities]
     List a directory
     [Word Processing]
 F10=Actions  Shift+F9=Command Prompt                             11:25a
```

Figure 2-54 —Active Task List in Program/File List View.

If the current view is Program List, then the screen will be split vertically, with the Program List appearing on the left and the Active Task List appearing on the right. Since tasks will usually be added from the Program List, it makes sense that the Active Task List appears in these two views.

Adding and Removing Programs in the Active Task List

The Active Task List shown in Figure 2-54 is empty, because no programs are running under Task Swapper. With Task Swapper enabled, running a program automatically places it in the Active Task List. Some programs will not work properly when you attempt to switch to or from them using the Task Swapper. If you want to run such programs with the Task Swapper enabled, they should be put into a Shell program in the Program List, and only run from there. This is because you can disable Task Swapping only for Shell programs. The technique for doing this was discussed earlier in this chapter, in the section titled, "Preventing the Task Swapper from Switching a Program."

There are three ways to add programs to the Active Task List:

1. Select the program in either the File List or Program List, and hold down ⇧ Shift and press Enter ↵; or put the mouse pointer on the program name, hold down ⇧ Shift, and double click. Either of these techniques will add the program to the Active Task List, without actually running it. This allows you to quickly add a program without having to wait for the Shell to swap out to it. Now when you want the program, you can simply go to it using the techniques described in the next section.

2. Run a program from the Program List, by selecting it with the arrow keys and pressing Enter ↵ or by double clicking on it with the mouse. The program will be started and you can work with it as you normally do. When you return to the Shell using the techniques described in the next section (Navigating in the Task Swapper), the program will be on the Active Task List.

3. Run a program from the File List, by selecting it, running it, and returning to the shell as described in step 2.

To remove a program from the Active Task List, terminate the program using its normal quit routine(s). The Shell will appear on the screen, and the program will no longer be in the Active Task List. If a program has locked up and you cannot terminate it normally, refer to the following section titled, "Handling Programs that Lock Up."

Navigating in the Task Swapper

When multiple programs are running under the Task Swapper, there are several ways of moving between the programs, and moving to and from the Shell.

To go directly to the Shell from any program, hold down Ctrl and press esc. Once in the Shell, you can switch to any other program by selecting it in the Active Task List and pressing Enter←, or by double clicking the mouse pointer on it in the Active Task List.

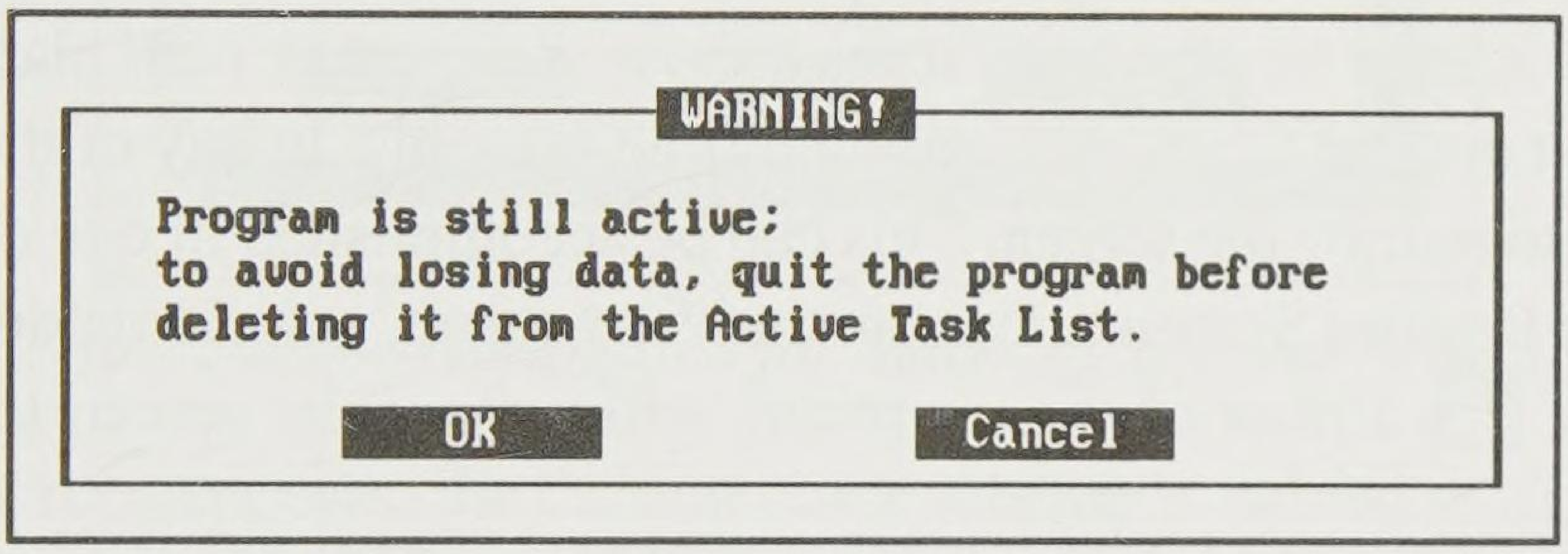
If an Application Shortcut Key was defined for the program, as described in the section "Defining An Application Shortcut Key," then pressing that key will take you to the program. Application shortcut keys can be pressed at any time — you could be in the Shell, or in another program in the Active Task List. However, the program you are switching to must have already been added to the Active Task List. If a program has not yet been placed in the Active Task List, its Shortcut Key will have no effect.

The last way to switch between programs is to hold down alt and repeatedly press tab. The first time you press tab, the screen will go blank except for a bar across the top, which will display the name of the next program in the Active Task List. Pressing tab again — while continuing to hold alt down — will display the next task in the list, and so on. When the name of the program you want to switch to appears in the bar at the top of the screen, release alt and the program will be activated.

The fastest ways to switch between two programs are to use Application Shortcut Keys, or the alt-tab combination. These are fastest because they do not actually load any intermediate programs. Going through the Shell and selecting the next program from the Active Task List involves swapping the current program out to disk, loading the Shell, then swapping the Shell out and loading the next program. Because of the amount of disk access this involves, it is substantially slower than going directly to the program.

Handling Programs that Lock Up

Some programs do not behave properly when you attempt to switch to or from them. In some cases, they will cause the computer to lock up — the keyboard will not respond and it is not possible to terminate the program. When this happens, it may still be possible to switch back to the Shell by holding down Ctrl and pressing esc. If this works, the program can be terminated by going to the Active Task List, selecting the program, and choosing Delete on the File menu. The dialog box shown in Figure 2-55 will be displayed.



```
Program is still active;
to avoid losing data, quit the program before
deleting it from the Active Task List.
```

Figure 2-55 — Dialog Box for Deleting an Active Task with the DELETE Option.

This option should only be used when there is no other way to terminate the program. After deleting the offending program, you should immediately quit all other programs, exit the Shell, and reboot your computer. A program that has locked up may have corrupted parts of DOS or other programs that are stored in memory. Continuing to run your programs may result in lost data or damaged files.

It may not be possible to return to the Shell when the computer has locked up. In this case, you will have to reboot your computer without returning to the Shell. Rebooting can be accomplished by simultaneously pressing Ctrl-alt-delete. If the computer does not reboot, you must press the Reset button, (if your computer has one), or turn off the power switch, wait about fifteen seconds, then turn it back on. If you have to reboot your computer while programs are running, you should check any files that were open at the time the machine locked up, in case they were damaged. Examples of files that might have been open are documents being edited in a word processor, a worksheet being used in a spreadsheet, or files that store data for a database.

Managing the Shell Environment

So far this chapter has focused on how to manipulate the various areas of the Shell screen, and how to use the specific features available in the Shell. There are some additional features that do not apply to a single area of the screen or a specific feature, but rather affect the overall appearance and performance of the Shell.

Redrawing and Updating the Screen

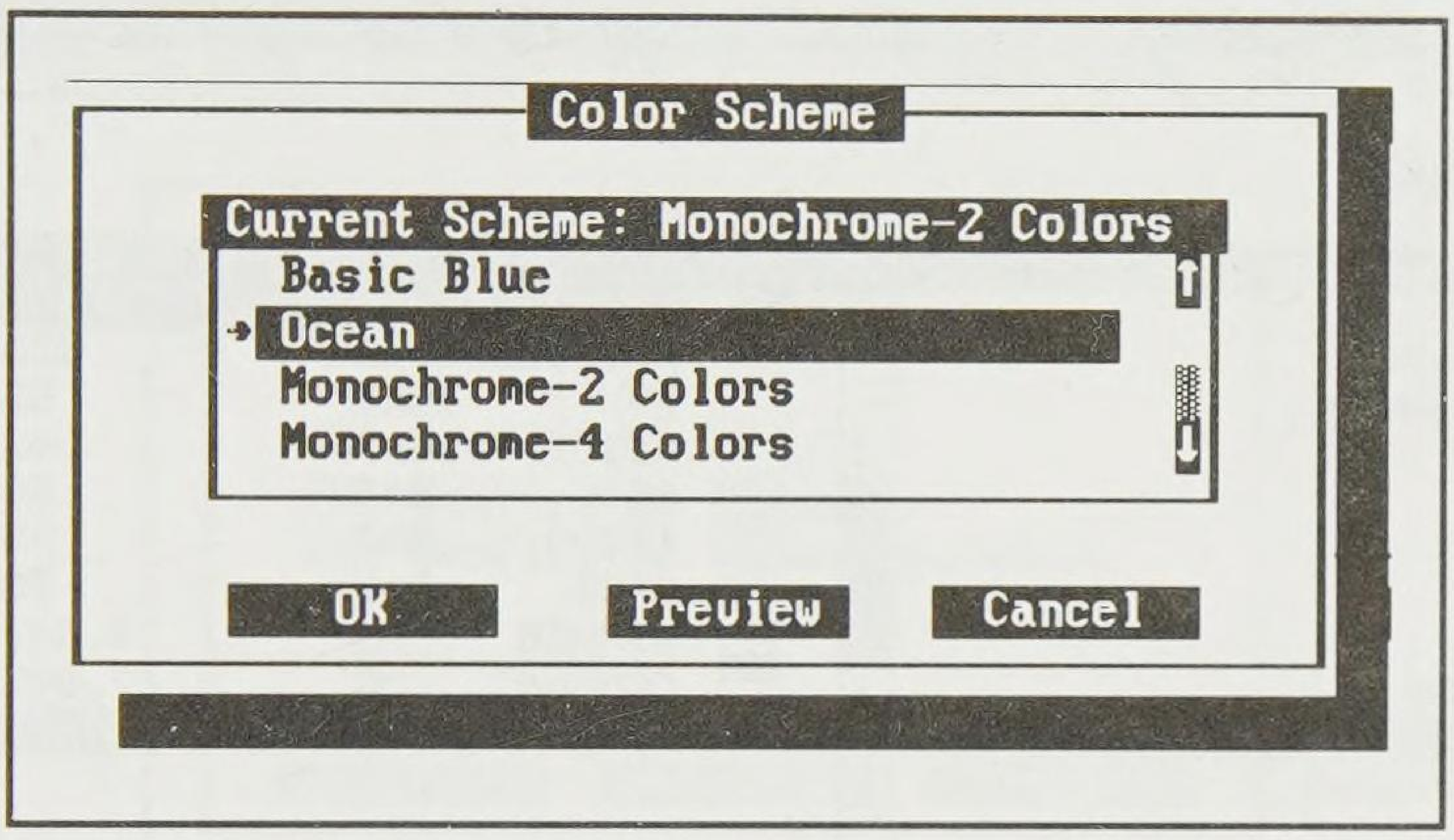
Under some circumstances you may find that the Shell screen has become corrupted — it may be all blank, it may show some areas with blank pieces in between, or it may be filled with unintelligible symbols. In any of these cases, it is necessary to redraw the screen. This can be accomplished in one of two ways: choosing the Repaint Screen option on the View menu, or holding down ⇧ Shift and pressing F5. Either of these options will restore the screen to its correct appearance.

When the Shell starts, it reads all of the directory and file names on the disk and constructs the Directory Tree and File List for the disk. If a directory or file is added or deleted by a program that is running under the Shell, the change will not be reflected in the Directory Tree and/or File List area, since it happened after those areas were constructed. To bring the Directory Tree and File List up to date, you must force the Shell to reread the contents of the disk. This can be done either by choosing the Refresh option from the View menu, or by pressing F5.

Changing Screen Colors

The Shell provides a number of different color schemes from which you may select. Schemes are complete color sets that define colors for all areas of the screen. While it is easy to switch schemes, the Shell does not provide a direct way of manipulating the colors of individual screen areas (the menu bar, or the drive box, for example). However, you can gain a great deal of fine control over the colors for different screen areas by editing the DOSSHELL.INI file. Techniques for doing this are discussed in the Advanced Topics section of this chapter. The Advanced Topics section should be read carefully before attempting to edit DOSSHELL.INI.

Several different color schemes are available from within the Shell. To change colors, select the Colors option on the Options menu. The Color Scheme dialog box, shown in Figure 2-56 will be displayed.



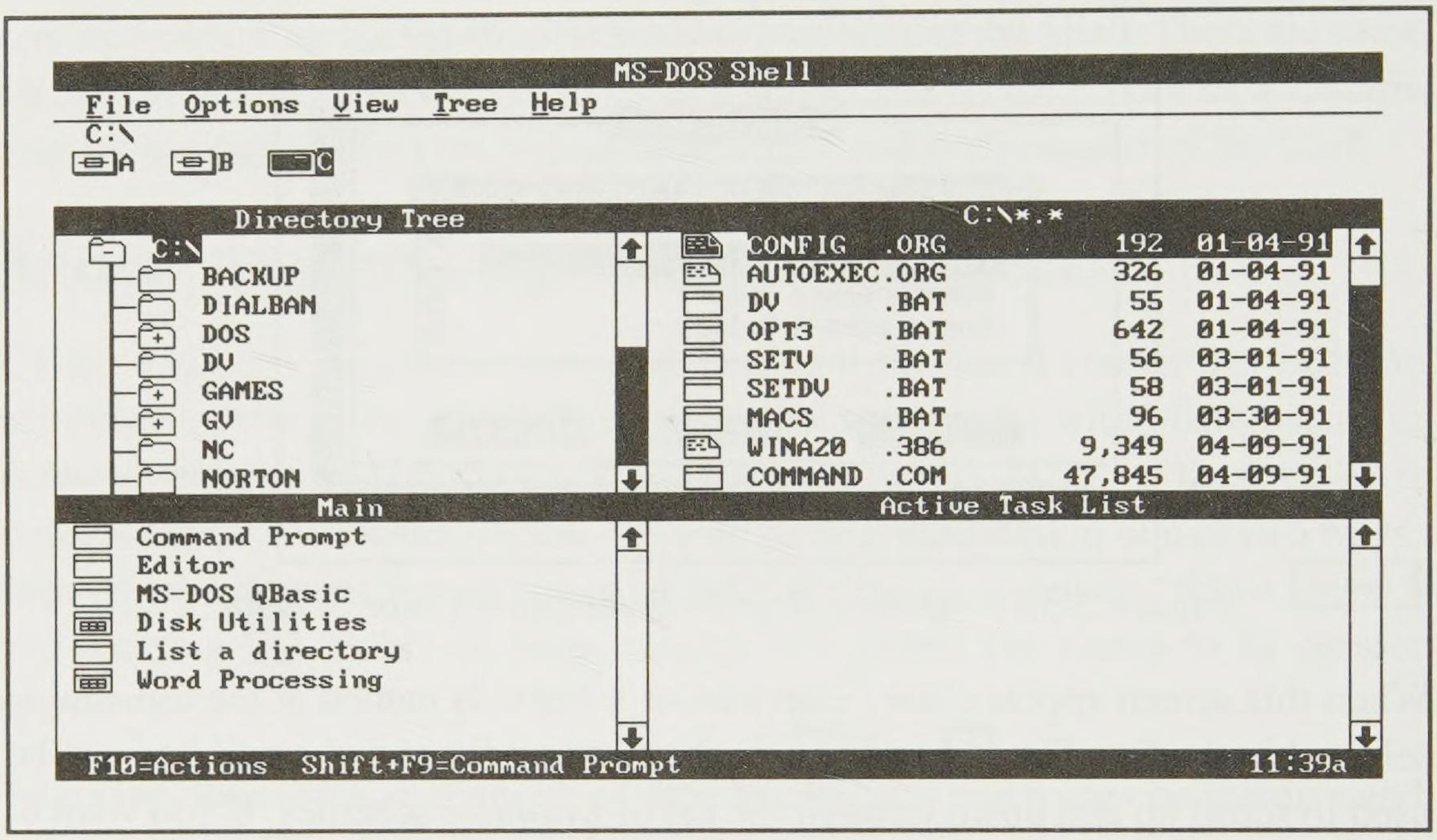
Current Scheme: Monochrome-2 Colors

Figure 2-56 — Dialog Box for Selecting a Color Scheme.

When this screen appears, the current color scheme is named at the top, and is selected in the list. The ↑ and ↓, or the mouse pointer and scroll bar, can be used to scroll up and down through the list of available schemes. If you want to see what a scheme is like before choosing it, select the scheme, then select the Preview button. The Color Scheme dialog box will remain on the screen, but the screen colors will change to reflect the new choice. If you like the new scheme, select OK or press Enter←. If you want to try another one, repeat this process. If, in the end, you decide you liked the original, you may choose Cancel and the original color scheme will be restored.

Text and Graphics Modes

If you have a graphics adapter card in your computer, you may choose to display the Shell screens in graphics mode rather than text mode (all of the screens shown so far in this chapter are in text mode). Common graphics cards include CGA, Hercules, EGA, VGA, and XGA. All display adapters support the text mode, which is the mode used by the Shell the first time you start it. The Shell screen with the Program/File Lists view, in graphics mode, is shown in Figure 2-57.



```
C:\
C:\
  BACKUP
  DIALBAN
  DOS
  DV
  GAMES
  GV
  NC
  NORTON

C:\*.*
CONFIG  .ORG       192  01-04-91
AUTOEXEC.ORG       326  01-04-91
DV      .BAT        55  01-04-91
OPT3    .BAT       642  01-04-91
SETV    .BAT        56  03-01-91
SETDV   .BAT        58  03-01-91
MACS    .BAT        96  03-30-91
WINA20  .386     9,349  04-09-91
COMMAND .COM    47,845  04-09-91

Command Prompt
Editor
MS-DOS QBasic
Disk Utilities
List a directory
Word Processing

F10=Actions  Shift+F9=Command Prompt                    11:39a
```

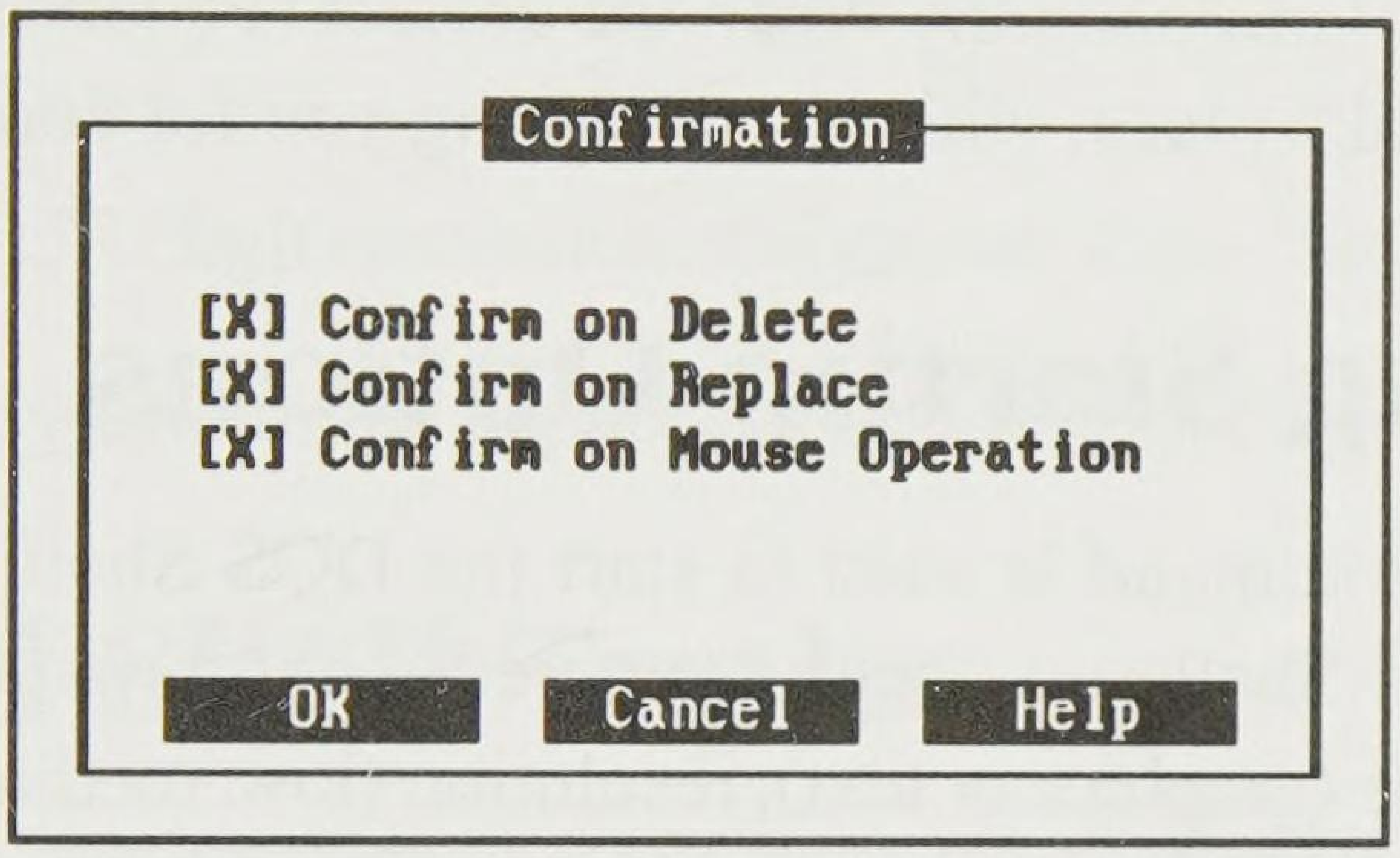
Figure 2-57 — Shell Screen in Graphics Mode.

The biggest difference between graphics mode and text mode is that the graphics mode uses icons, or small pictures, to indicate the type of object an item is. For example, in the File List area, data files do not have the same icon as executable files, and in the Program List area, groups have a different icon than programs.

Controlling the Display of Confirmation Dialog Boxes

Certain operations, such as deleting files and directories, cause confirmation dialog boxes to appear. Confirmation dialog boxes are safeguards that are built into the Shell to protect against accidentally deleting something that you need. When you are working on a long list of items, however, the confirmation boxes can be a nuisance, since you have to respond to them for each item. Consequently, the Shell provides a way to turn them off. This should always be done with caution, since it removes an important level of protection against mistakes. In general, it is wise to turn these off for a specific operation, then turn them back on again when you are finished. Remember, if they are off when you exit the Shell, they will be off when you run it again.

To turn confirmation messages on or off, select Confirmation from the Options menu. This brings up the File Options dialog box shown in Figure 2-58.



[X] Confirm on Delete
[X] Confirm on Replace
[X] Confirm on Mouse Operation

Figure 2-58 — Confirmation Options Dialog Box.

The default is for all of the options to be turned on, as indicated by the X's in the check boxes. Turning off an option is accomplished by selecting the option and pressing space, or clicking once on the option with the mouse pointer. The X will disappear. The same steps are used to turn the options back on again. When you have each option the way you want it, select the OK button or press Enter← to save your choices. Selecting the Cancel button or pressing esc will cancel your choices.

Confirm on Delete: Controls the Delete File Confirmation box (see Figure 2-31), and the corresponding confirmation box for deleting directories. With this option turned off, files and directories are deleted automatically when they are selected and delete is pressed, or when Delete is chosen from the File menu. You will not be prompted to confirm the deletion.

Confirm on Replace: Controls the Replace File Confirmation dialog box (see Figure 2-27), which is displayed when you attempt to a copy or move a file to another directory which already has a file with the same name in it. With this option turned off, the file in the target directory will automatically be replaced with the file in the source directory when Copy or Move is chosen from the File menu. You will not be prompted to confirm the replacement.

Confirm on Mouse Operation: Controls the Confirm Mouse Operation dialog box. Certain operations can be accomplished by mouse movement without going through the menus — for example, copying and moving files between directories. When these operations are attempted, the Confirm Mouse Operation dialog box (shown in Figure 2-25) is displayed. When this option is turned off, these mouse-driven events occur automatically when the correct sequences of keystrokes and mouse operations take place, without prompting you for confirmation.

Specifying Startup Options

The DOSSHELL command is used to start the DOS Shell from the command prompt. If the DOS Shell command is entered with no parameters, it starts the Shell using the mode (graphics or text), resolution (low, medium, or high), number of lines per inch, and color setup, last used. However, you may override some of these defaults with the following switches.

/B is used to start the Shell in black and white mode. For example:

```
DOSSHELL /B
```

/G is used to start the Shell in graphics mode. If your monitor supports more than one resolution, you may follow the G with a colon and the letter L, M, or H, for Low, Medium, and High resolutions. For example, the following command starts the Shell in High resolution graphics mode:

```
DOSSHELL /G:H
```

/T is used to specify text mode. As with the /G switch, a resolution specifier may be included. The following command starts the Shell in Low resolution text mode:

```
DOSSHELL /T:L
```

In addition to Low, Medium, and High resolutions, there are sub-categories that indicate how many lines of text should be displayed on the screen. If you examine the Screen Display Mode dialog box (Figure 2-4), you will notice that on the right side of some of the choices is a number. For example, the "Text 43 lines High Resolution 1" line has a 1 at its right edge.

The number at the right edge is a code for the number of lines per inch, at the specified resolution. This code may also be specified with the /G and /T switches, if the resolution is specified as well. The following commands illustrate this:

```
DOSSHELL /G:M2
```

starts the Shell in Medium resolution graphics mode, at 34 lines per inch

```
DOSSHELL /T:H1
```

starts the Shell in High resolution text mode, at 43 lines per inch.

These settings correspond to the capabilities of a VGA color monitor.

Advanced Shell Topics

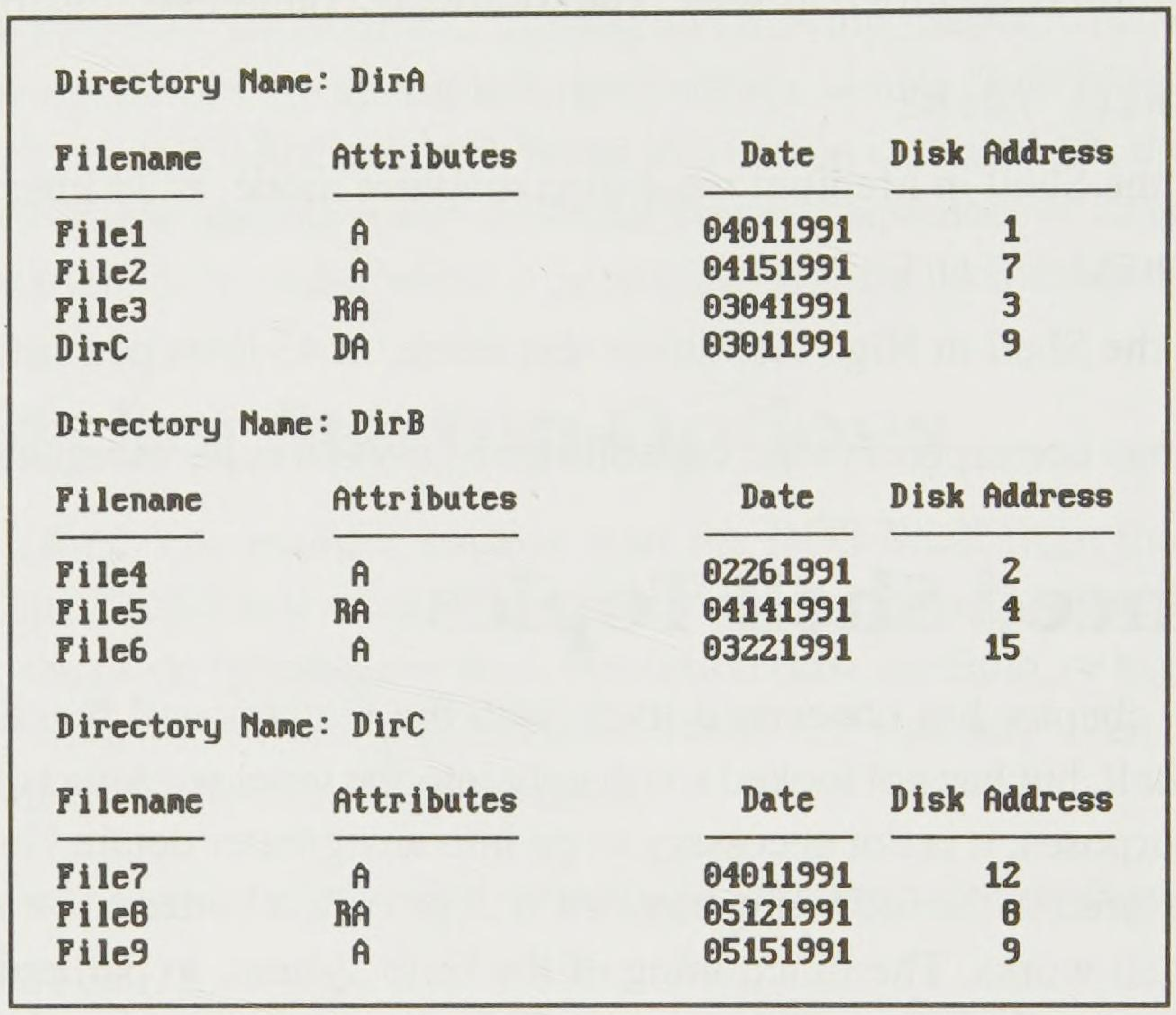
So far, this chapter has concerned itself with the features and functions of the DOS 5.0 Shell, but has not looked too deeply into the inner workings of the Shell. For most purposes, it is not necessary to go into any greater detail. However, the material covered in the following sections will provide a better understanding of how the Shell works. The functioning of the Help system, in particular, will be examined, and techniques will be presented for greatly enhancing the degree of online help you can add to your own Shell programs. Techniques for giving you fine control over the Shell colors will also be reviewed.

Understanding the Move File Operation

In this chapter, an earlier section titled, “Moving Files from One Directory to Another,” stated that the Move operation is faster and better than performing Copy and Delete operations. The reason for this is based on the way DOS manages directories and files on a disk.

Directories are just special types of files. What is special about them is that they have their Directory attribute set to on, so DOS knows they are directories. They also have a particular structure for keeping track of information about the files they contain — their names, date and time of last modification, their attributes, and information about their physical location on the disk.

Figure 2-59 shows a schematic representation of three directories, named DirA, DirB, and DirC. DirC is a subdirectory of directory A.



Directory Name: DirA

Filename	Attributes	Date	Disk Address
File1	A	04011991	1
File2	A	04151991	7
File3	RA	03041991	3
DirC	DA	03011991	9

Directory Name: DirB

Filename	Attributes	Date	Disk Address
File4	A	02261991	2
File5	RA	04141991	4
File6	A	03221991	15

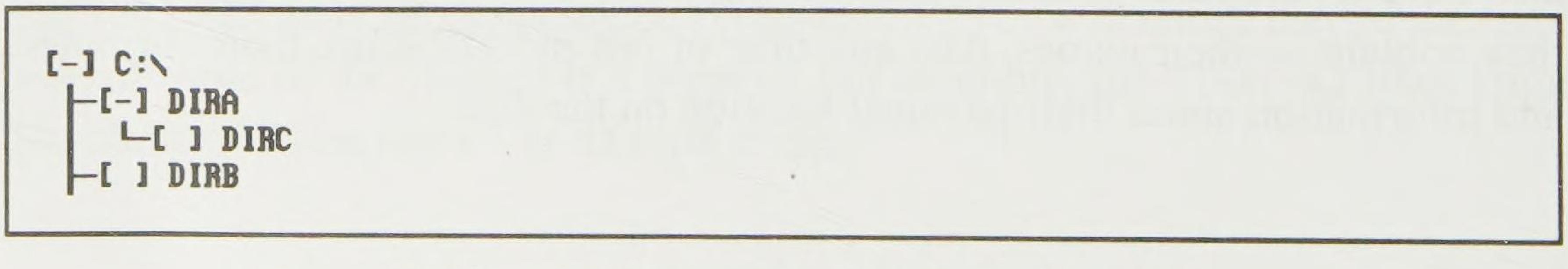
Directory Name: DirC

Filename	Attributes	Date	Disk Address
File7	A	04011991	12
File8	RA	05121991	8
File9	A	05151991	9

Figure 2-59 — Schematic Representation of Three Directories.

The physical location of a file on the disk has nothing to do with which directory it is in. When a file is created in a particular directory, the file itself is placed wherever there is free space on the disk. Within the directory, the location of the file is recorded. This provides for a very flexible system — a new file can be added to whatever directory you like, as long as there is room somewhere on the disk for the file.

When the Shell starts up, it reads all the directories on the disk and constructs the graphical tree for the Directory Tree Area, and the File Lists that will appear for each directory. The Directory Tree for the directories in Figure 2-59 is shown in Figure 2-60.



```
[-] C:\
├─[-] DIRA
│   └─[ ] DIRC
└─[ ] DIRB
```

Figure 2-60 — Directory Tree for Sample Directories.

From this discussion, it is evident that the only thing that determines which directory a particular file resides in, is the directory that contains the information for that file. So to move File9 from DirC to DirB simply involves copying the directory entry for File9 from DirC to DirB, then deleting it from DirC. Consequently, moving a file really only involves copying this small amount of information from one directory to another, then removing it from the first directory. This is what the Move option accomplishes. For large files it saves considerable time over copying and deleting. It also creates less fragmentation on your disk and reduces wear and tear on the drive.

This also explains why a Rename operation is simple and fast — all that is changed is the name of the file in its directory.

Structure of the DOSSHELL.INI File

There is a file named DOSSHELL.INI in the DOS directory. DOSSHELL.INI contains all of the initialization information for the Shell. The Shell reads this file each time it starts, to determine the current Shell configuration. Whenever you make a change to the Shell configuration, such as going from text to graphics mode, changing the color scheme, or adding a program or group in the Program List area, the DOSSHELL.INI file is updated accordingly. This is how the Shell manages to always bring back the same setup you last used. If you know what information is stored in DOSSHELL.INI, and how it is structured, you can manipulate it to gain greater flexibility and control over the Shell colors. You can also provide more online help for your programs than the Shell itself allows. A typical copy of DOSSHELL.INI appears in Listing 2-1.

STOP

Note: This figure contains line numbers and wrapped lines. These are for illustrative purposes only. To properly function, DOSSHELL.INI cannot have line numbers and all of the text for a command must be on a single line — it cannot be wrapped to multiple lines.

Listing 2-1 — Sample DOSSHELL.INI File.

```
****************  WARNING  ********************
This file may contain lines with more than 256
characters. Some editors will truncate or split
these lines. If you are not sure whether your
editor can handle long lines, exit now without
saving the file.

Note: The editor which is invoked by the
      MS-DOS 5.0 EDIT command can be used
     to edit this file.
****************  NOTE  ***********************
Everything up to the first left square bracket
character is considered a comment.
***********************************************

**********  ADDITIONAL COMMENTS  **************

This file has been modified to contain a custom color group,
named RAINBOW.

*************************************************

[savestate]
screenmode = text
resolution = low
startup = filemanager
filemanagermode = shared
sortkey = date
pause = disabled
explicitselection = disabled
swapmouse = disabled
tasklist = disabled
switching = disabled
mouseinfo =
sortorder = descending
displayhiddenfiles = disabled
replaceconfirm = enabled
deleteconfirm = enabled
mouseconfirm = enabled
crossdirselection = disabled
[programstarter]
currentcolor = Rainbow
filemanager = enabled
command = enabled
group =
```

continued…

...from previous page (Listing 2-1).

```
{
    program =
    {
        command = COMMAND
        title = Command Prompt
        help = Starts the MS-DOS command prompt where you can
               type any MS-DOS command.^m^mTo return to MS-DOS
               Shell from the command line:^m^m1. Type exit^m2.
               Press ENTER.^m^mRelated Topic^m   " More on MS-DOS
               Command Prompt "~S129~
        pause = disabled
    }
    program =
    {
        command = EDIT %1
        title = Editor
        help = Starts MS-DOS Editor, a text editor you can use to
               create and modify text files such as batch files.
               After you choose Editor, you can specify the file
               you want to work with in a dialog box.^m^mRelated
               Topic^m   " More on MS-DOS Editor "~S130~
        pause = disabled
        dialog =
        {
            title = File to Edit
            info = Enter the name of the file to edit. To start
                   MS-DOS Editor without opening a file, delete
                   the suggestion and then press ENTER.
            prompt = File to edit?
            parameter = %1
            default = %l
        }
        screenmode = text
        alttab = enabled
        altesc = enabled
        ctrlesc = enabled
        prevent = disabled
    }
    program =
    {
        command = QBASIC %1
        title = MS-DOS QBasic
        help = Starts MS-DOS QBasic, a programming environment
               you can use to create, modify, run, and debug
               programs. After you choose MS-DOS QBasic, you can
               specify the file you want to work with in a dialog
```

continued...

...from previous page (Listing 2-1).

```
           box.^m^mRelated Topic^m   " More on MS-DOS QBasic
           "~S131~
    pause = disabled
    dialog =
    {
        title = MS-DOS QBasic File
        info = Enter the name of a QBasic program. To start
               MS-DOS QBasic without opening a program,
               delete the suggestion and then press ENTER.
        prompt = QBasic File?
        parameter = %1
    }
}
group =
{
    title = Disk Utilities
    help = Displays program items you can choose to manage
           your disks. You can also choose to open the Main
           group or any group you may have added.
    program =
    {
        command = diskcopy %1
        title = Disk Copy
        pause = enabled
        dialog =
        {
            title = Disk Copy
            info = Enter the source and destination drives.
            prompt = Parameters . . .
            default = a: b:
            parameter = %1
        }
        help = Temporarily leaves MS-DOS Shell to copy the
               contents of a floppy disk to another floppy
               disk. After you choose Disk Copy, a dialog box
               suggests parameters and switches you can
               either accept or replace.^m^mRelated Topic^m
               " More on Disk Copy "~S132~
    }
    program =
    {
        command = backup %1
        title = Backup Fixed Disk
        pause = enabled
        dialog =
```

continued...

...from previous page (Listing 2-1).

```
            {
                title = Backup Fixed Disk
                info = Enter the source and destination drives.
                prompt = Parameters . . .
                default = c:\*.* a: /s
                parameter = %1
            }
            help = Temporarily leaves MS-DOS Shell to copy files
                   from your hard disk to another disk. After you
                   choose Backup Fixed Disk, a dialog box
                   suggests parameters and switches you can
                   either accept or replace.^m^mRelated Topic^m
                   " More on Backup Fixed Disk "~S133~
        }
        program =
        {
            command = restore %1
            title = Restore Fixed Disk
            pause = enabled
            dialog =
            {
                title = Restore Fixed Disk
                info = Enter the source and destination drives.
                prompt = Parameters . . .
                parameter = %1
            }
            help = Temporarily leaves MS-DOS Shell to restore
                   files that were backed up. After you choose
                   Restore Fixed Disk, a dialog box suggests
                   parameters and switches you can either accept
                   or replace.^m^mRelated Topic^m    " More on
                   Restore Fixed Disk "~S134~
        }
        program =
        {
            command = format %1 /q
            title = Quick Format
            pause = enabled
            dialog =
            {
                title = Quick Format
                info = Enter the drive to quick format.
                prompt = Parameters . . .
                default = a:
                parameter = %1
            }
```

continued...

...from previous page (Listing 2-1).

```
        help = Temporarily leaves MS-DOS Shell to prepare a
               disk to accept MS-DOS files. After you choose
               Quick Format, a dialog box suggests parameters
               and switches you can either accept or
               replace.^m^mRelated Topic^m   " More on Quick
               Format "~S136~
        screenmode = text
        alttab = enabled
        altesc = enabled
        ctrlesc = enabled
        prevent = enabled
    }
    program =
    {
        command = format %1
        title = Format
        pause = enabled
        dialog =
        {
            title = Format
            info = Enter the drive to format.
            prompt = Parameters . . .
            default = a:
            parameter = %1
        }
        help = Temporarily leaves MS-DOS Shell to prepare a
               disk to accept MS-DOS files. After you choose
               Format, a dialog box suggests parameters and
               switches you can either accept or
               replace.^m^mRelated Topic^m   " More on Format
               "~S135~
    }
    program =
    {
        command = undelete %1
        title = Undelete
        help = Recovers deleted files. WARNING: If your disk
               is full or if you are using task swapping,
               using this program item may render some
               deleted files unrecoverable.^m^mRelated
               Topic^m   " More on Undelete "~S137~
        pause = enabled
        dialog =
        {
            title = Undelete
```

continued...

...from previous page (Listing 2-1).

```
                info = WARNING! This action may cause the
                       permanent loss of some deleted files.
                       Press F1 for more information.
                prompt = Parameters . . .
                default = /LIST
                parameter = %1
            }
            screenmode = text
            alttab = enabled
            altesc = enabled
            ctrlesc = enabled
            prevent = enabled
        }
    }
    group =
    {
        program =
        {
            command = dir c:\dv\temp ; dir c:\dv\temp ; IF %1 ==
                      WP CALL WP ; IF %1 == wp CALL WP ; IF %1 ==
                      SS CALL SS ; IF %1 == ss CALL SS
            title = Test for batch calls.
            pause = enabled
            dialog =
            {
                title = Selection After EMail
                info = Enter WP to go to Word Processing, or SS
                       to go to the Spreadsheet.  Anything else
                       is ignored.
                prompt = Program choice
                parameter = %1
            }
            dialog =
            {
                title = Delete Verification
                info = Enter Y to delete the files from
                       c:\dv\temp.  Enter any other letter to not
                       delete them.
                prompt = Delete (Y/N)?
                default = N
                parameter = %2
            }
            screenmode = text
            alttab = enabled
            altesc = enabled
            ctrlesc = enabled
```

continued...

...from previous page (Listing 2-1).

```
            prevent = enabled
        }
        program =
        {
            command = if %1 == Y echo hello there you all
            title = echo tester
            pause = enabled
            dialog =
            {
                title = Echo Verification
                info = Enter Y to verify that the message should
                       be echoed.
                prompt = Echo (Y/N)?
                parameter = %1
            }
        }
        program =
        {
            command = wp
            title = Word Processing
            pause = enabled
        }
        title = Word Processing
        help = Choose this option to use the word processor.
        program =
        {
            command = Dir
            title = Test program 1
            pause = enabled
        }
    }
    special = default
}
color =
{
    selection =
    {
        title = Basic Blue
        foreground =
        {
            base = black
            highlight = brightwhite
            selection = brightwhite
            alert = brightred
            menubar = black
```

continued...

...from previous page (Listing 2-1).

```
            menu = black
            disabled = white
            accelerator = cyan
            dialog = black
            button = black
            elevator = white
            titlebar = black
            scrollbar = brightwhite
            borders = black
            drivebox = black
            driveicon = black
            cursor = black
        }
        background =
        {
            base = brightwhite
            highlight = blue
            selection = black
            alert = brightwhite
            menubar = brightwhite
            menu = brightwhite
            disabled = brightwhite
            accelerator = brightwhite
            dialog = brightwhite
            button = white
            elevator = white
            titlebar = white
            scrollbar = black
            borders = brightwhite
            drivebox = brightwhite
            driveicon = brightwhite
            cursor = brightcyan
        }
    }
    selection =
    {
        title = Ocean
        foreground =
        {
            base = black
            highlight = brightwhite
            selection = brightwhite
            alert = brightred
            menubar = black
            menu = black
            disabled = white\
```

continued...

...from previous page (Listing 2-1).

```
            accelerator = brightwhite
            dialog = black
            button = black
            elevator = white
            titlebar = black
            scrollbar = brightwhite
            borders = black
            drivebox = black
            driveicon = black
            cursor = black
        }
        background =
        {
            base = brightwhite
            highlight = blue
            selection = black
            alert = brightwhite
            menubar = brightgreen
            menu = cyan
            disabled = cyan
            accelerator = cyan
            dialog = cyan
            button = brightwhite
            elevator = white
            titlebar = white
            scrollbar = black
            borders = black
            drivebox = brightwhite
            driveicon = brightwhite
            cursor = brightgreen
        }
    }
    selection =
    {
        title = Monochrome-2 Colors
        foreground =
        {
            base = black
            highlight = white
            selection = white
            alert = white
            menubar = black
            menu = black
            disabled = white
            accelerator = white
```

continued...

...from previous page (Listing 2-1).

```
            dialog = black
            button = white
            elevator = black
            titlebar = white
            scrollbar = white
            borders = black
            drivebox = black
            driveicon = black
        }
        background =
        {
            base = white
            highlight = black
            selection = black
            alert = black
            menubar = white
            menu = white
            disabled = white
            accelerator = black
            dialog = white
            button = black
            elevator = white
            titlebar = black
            scrollbar = black
            borders = black
            drivebox = white
            driveicon = white
        }
    }
    selection =
    {
        title = Monochrome-4 Colors
        foreground =
        {
            base = black
            highlight = brightwhite
            selection = brightwhite
            alert = black
            menubar = black
            menu = black
            disabled = white
            accelerator = brightwhite
            dialog = black
            button = black
            elevator = white
            titlebar = black
```

continued...

...from previous page (Listing 2-1).

```
            scrollbar = brightwhite
            borders = black
            drivebox = black
            driveicon = black
        }
        background =
        {
            base = brightwhite
            highlight = brightblack
            selection = brightblack
            alert = brightwhite
            menubar = brightwhite
            menu = brightwhite
            disabled = brightwhite
            accelerator = brightblack
            dialog = brightwhite
            button = white
            elevator = white
            titlebar = white
            scrollbar = black
            borders = black
            drivebox = brightwhite
            driveicon = brightwhite
        }
    }
    selection =
    {
        title = Reverse
        foreground =
        {
            base = white
            highlight = black
            selection = black
            alert = black
            menubar = white
            menu = white
            disabled = black
            accelerator = black
            dialog = white
            button = black
            elevator = white
            titlebar = black
            scrollbar = black
            borders = white
            drivebox = white
```

continued...

...from previous page (Listing 2-1).

```
            driveicon = white
        }
        background =
        {
            base = black
            highlight = white
            selection = white
            alert = white
            menubar = black
            menu = black
            disabled = black
            accelerator = white
            dialog = black
            button = white
            elevator = black
            titlebar = white
            scrollbar = white
            borders = black
            drivebox = black
            driveicon = black
        }
    }
    selection =
    {
        title = Hot Pink
        foreground =
        {
            base = black
            highlight = brightwhite
            selection = brightwhite
            alert = brightmagenta
            menubar = black
            menu = black
            disabled = white
            accelerator = magenta
            dialog = black
            button = brightwhite
            elevator = white
            titlebar = brightwhite
            scrollbar = brightwhite
            borders = black
            drivebox = black
            driveicon = black
            cursor = brightwhite
        }
        background =
```

continued...

...from previous page (Listing 2-1).

```
        {
            base = brightwhite
            highlight = brightmagenta
            selection = magenta
            alert = brightwhite
            menubar = brightwhite
            menu = brightwhite
            disabled = brightwhite
            accelerator = brightwhite
            dialog = brightwhite
            button = magenta
            elevator = white
            titlebar = magenta
            scrollbar = black
            borders = black
            drivebox = brightwhite
            driveicon = brightwhite
            cursor = brightred
        }
    }
    selection =
    {
        title = Emerald City
        foreground =
        {
            base = black
            highlight = black
            selection = brightwhite
            alert = brightgreen
            menubar = black
            menu = black
            disabled = white
            accelerator = green
            dialog = black
            button = brightwhite
            elevator = white
            titlebar = brightwhite
            scrollbar = brightwhite
            borders = black
            drivebox = black
            driveicon = black
            cursor = black
        }
        background =
        {
```

continued...

…from previous page (Listing 2-1).

```
            base = brightwhite
            highlight = brightgreen
            selection = green
            alert = brightwhite
            menubar = brightwhite
            menu = brightwhite
            disabled = brightwhite
            accelerator = brightwhite
            dialog = brightwhite
            button = green
            elevator = white
            titlebar = green
            scrollbar = black
            borders = black
            drivebox = brightwhite
            driveicon = brightwhite
            cursor = brightcyan
        }
    }
    selection =
    {
        title = Turquoise
        foreground =
        {
            base = black
            highlight = brightwhite
            selection = brightwhite
            alert = brightred
            menubar = brightwhite
            menu = black
            disabled = white
            accelerator = brightwhite
            dialog = black
            button = black
            elevator = white
            titlebar = black
            scrollbar = brightwhite
            borders = black
            drivebox = black
            driveicon = black
            cursor = black
        }
        background =
        {
            base = brightwhite
            highlight = brightblue
```

continued…

...from previous page (Listing 2-1).

```
            selection = black
            alert = brightwhite
            menubar = brightcyan
            menu = brightcyan
            disabled = brightcyan
            accelerator = brightcyan
            dialog = brightcyan
            button = brightwhite
            elevator = white
            titlebar = white
            scrollbar = black
            borders = black
            drivebox = brightwhite
            driveicon = brightwhite
            cursor = brightgreen
        }
    }
    selection =
    {
        title = Rainbow
        foreground =
        {
            base = brightblack
            highlight = black
            selection = brightmagenta
            alert = brightred
            menubar = brightwhite
            menu = brightgreen
            disabled = black
            accelerator = brightred
            dialog = brightcyan
            button = magenta
            elevator = brightgreen
            titlebar = red
            scrollbar = cyan
            borders = brightgreen
            drivebox = brightyellow
            driveicon = brightcyan
            cursor = black
        }
        background =
        {
            base = brightmagenta
            highlight = green
            selection = black
```

continued...

...from previous page (Listing 2-1).

```
            alert = white
            menubar = brightwhite
            menu = brightblue
            disabled = brightcyan
            accelerator = brightcyan
            dialog = blue
            button = brightwhite
            elevator = brightgreen
            titlebar = brightyellow
            scrollbar = blue
            borders = brightgreen
            drivebox = brown
            driveicon = cyan
            cursor = magenta
        }
    }
}
associations =
{
    association =
    {
        program = EDIT
        extension = TXT
    }
    association =
    {
        program = QBASIC /run
        extension = BAS
    }
}
```

There are some very long `command=`, `info=`, and `help=` lines in DOSSHELL.INI. For illustrative purposes, these have been wrapped to the next line, but you must be certain, as discussed below, that you do not save a DOSSHELL.INI file with wrapped lines.

DOSSHELL.INI is an ASCII text file, which means it can be examined and modified by a text editor. You should not modify it unless you are absolutely certain of what you are doing. Before modifying it, you should create a copy of it with the DOS copy command, or the COPY feature on the File menu of the Shell. That way you can always restore the original from the copy if you make any mistakes you can't undo. When you save the file, make certain that you save it as an ASCII file or a DOS text file. Otherwise your word processor may put special codes into the file that will make it unusable by the Shell. If you use the EDIT program that comes with DOS 5.0 you won't need to worry about how you save the file — it only works in ASCII format.

DOSSHELL.INI may contain lines that are longer than 256 characters. Some word processors will either split these lines or truncate them. You must be certain your word processor can manage lines longer than 256 characters long before using it to edit this file. The EDIT editor that comes with DOS can handle lines more than 256 characters long.

WARNING

You must completely exit the Shell BEFORE editing DOSSHELL.INI because DOSSHELL.INI is written to the disk each time you exit the Shell. If you edit the file while the Shell is running, the edited file will be overwritten when you exit the Shell, eliminating any changes you have made.

When you examine DOSSHELL.INI, you may see that it begins with some comments, either from Microsoft or from the vendor that sold you DOS 5.0. You may add your own comments as well, to document any changes you make to the file. Following the comments is the line:

```
[savestate]
```

Everything up to this line is treated as a comment — the Shell program ignores it. This is, therefore, a good place to store explanatory comments for anyone that might examine the file in the future. In Listing 2-1, lines 1 through 22 are comments.

There are two main sections in the DOSSHELL.INI file, labeled `[savestate]` and `[programstarter]`. Lines 23 through 40 constitute the `[savestate]` section of DOSSHELL.INI. This section contains the settings for global switches, such as the screen mode (text versus graphics), and the confirmation switches (such as confirm on replace). These are all readily changeable within the Shell, so they should be left alone in DOSSHELL.INI.

What is of real interest is the `[programstarter]` section, which contains information about the screen contents. This is everything from line 41 to the end of the file. There are a few switch settings at the beginning of this section — lines 42, 43, and 44. Following these switch settings are the definitions of all the groups and programs that will display in the Program List area. This is the area in which you can add hypertext help to your own programs. The group and program definitions go from line 45 through line 406.

Following the program and group definitions is the color definition area, which defines the specific colors for each section of the screen, and each color scheme. The color scheme definition area goes from line 307 to line 699.

Following the color scheme definition is the associations definition area, which defines any associations you have set up between file extensions and programs. The association definition part of the file does not need to be manipulated because it is easier and safer to manage these associations from within the Shell. The sections that provide opportunities for expanding the Shell capabilities are the group/program definitions and the color scheme definitions.

The DOSSHELL.INI file can be copied from one computer to another. This is a quick and easy way to duplicate a setup on multiple machines. However, you should make sure the second machine has the necessary files and directories to support any Shell programs you've written. If you have modified the DOSSHELL.HLP file, then that, too, should be copied over.

Developing Advanced Help Screens

Many of the Shell's built-in help screens contain references to related topics or procedures. If one of these references is selected, a second help screen can be displayed that goes into more detail on the referenced topic. The second help screen may, in turn, contain references to further related topics that can be selected. When two or more help screens are linked together in this way, you can move backwards through the ones you have seen by choosing the Back option button on the Help dialog boxes.

An example of linked help screens can be seen by making the File List area active and pressing F1. This displays the dialog box shown in Figure 2-7. At the bottom of this box is the line "Next topic;" scrolling down a little will show the phrase "Disk Drives." Selecting this phrase and pressing Enter←, or double-clicking on it with the mouse, will display a second dialog box that goes into more detail on disk drives. This second box is linked to two more help boxes, "Selecting Drives" and "Directories." Selecting the Back button returns you to the previous help screen.

Linked screens provide a convenient way to create lengthy online help. They also let users select their own path through the screens, choosing just those topics that they need help with. However, the Shell does not allow you to create linked screens for your own programs. To accomplish this, the DOSSHELL.INI and DOSSHELL.HLP files must be edited.

Linked help screens are managed by two different files: the DOSSHELL.INI file, and the DOSSHELL.HLP file. As discussed above, the DOSSHELL.INI file is modified whenever you change some aspect of the Shell. DOSSHELL.HLP is not changed by the Shell.

When you add a program to the Shell, and provide online help for it, the help text is added to the DOSSHELL.INI file. Line 297 in Figure 2-61 is an example: it contains the text "Choose this option to use the word processor," which was entered as the help text for the Word Processor option that was added to the Program List. If you look at lines 62 through 66 in Figure 2-61, you will see some typical help text for one of the built-in programs in the Shell — the Editor program. (Remember that Figure 2-61 shows lines wrapped, but that in the actual DOSSHELL.INI file this text should all appear on one long line). The help dialog ends with the following text:

```
^m^mRelated Topic^m " More on MS-DOS Editor "~S130~
```

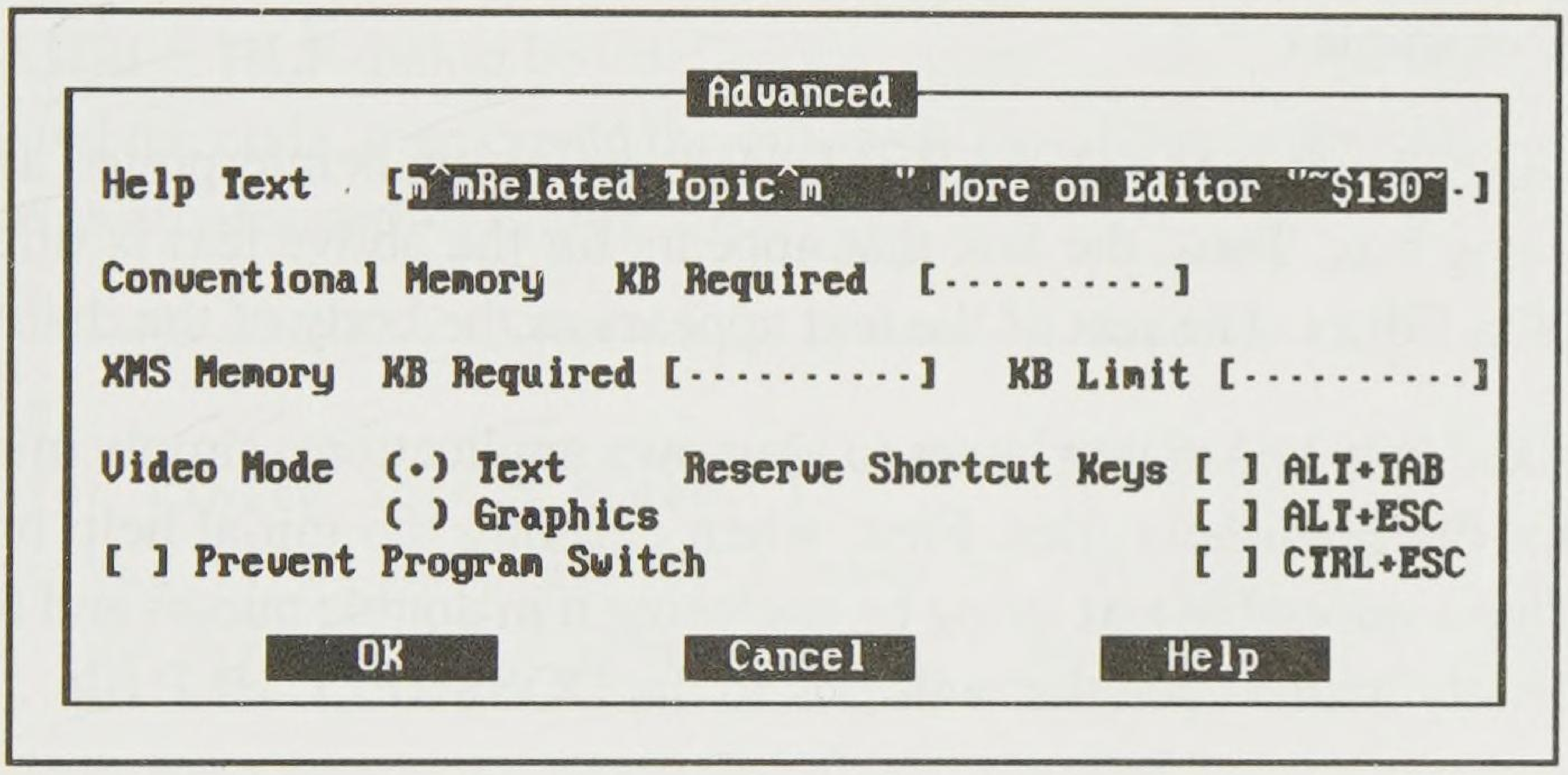
As discussed earlier in this chapter, the caret+m (^m) symbols tell the Shell to skip to the next line when displaying the text that follows the symbol.

At the end of the Editor help text is a code (S130), surrounded by tildes (~). The tilde symbols tell the Shell that the immediately preceding text, which is surrounded by double quotes, should be selectable, and that when it is selected and Enter← pressed, a second dialog box should appear that has more help text in it. The text for the second dialog box is labelled S130, and is located in the DOSSHELL.HLP file.

This can all be seen within the Shell, by examining the help text in the Advanced Properties dialog box for the Editor program. The following steps will display the Advanced Options dialog box:

1. Select Editor in the Program List
2. Press alt-F to pull down the File menu
3. Press P to bring up the Properties dialog box
4. Select the Advanced button and press Enter←.

After following these steps, the cursor will be at the beginning of the Help Text text box for the Editor program. Now press end and the cursor will move to the end of the Help text, where you will see the dialog box shown in Figure 2-61.



```
                         Advanced
Help Text  [m^mRelated Topic^m     " More on Editor "~S130~-]

Conventional Memory   KB Required  [..........]

XMS Memory  KB Required [..........]  KB Limit [..........]

Video Mode  (•) Text      Reserve Shortcut Keys [ ] ALT+TAB
            ( ) Graphics                        [ ] ALT+ESC
[ ] Prevent Program Switch                      [ ] CTRL+ESC

          OK               Cancel               Help
```

Figure 2-61 —Advanced Options Dialog Box
Showing End of Help Text for Editor.

At the end of the Help Text is the ~S130~ code. Notice that there is no gap between the final double quotes and the first tilde. If a space was left here, the entire text string — S130 code and all — would simply appear as part of the help text, and the Shell would not attempt to locate a linked help/dialog box in DOSSHELL.HLP.

Opening the DOSSHELL.HLP file and searching for the text string S130 shows the following block of text:

```
!S130!@
More on MS-DOS Editor
When you quit MS-DOS Editor, you will return to MS-DOS Shell.
--@
```

Note that there are no tildes here — if DOSSHELL.HLP were searched for the text string ~S130~, it would not be found. The tildes simply mark the pointer to a subsequent dialog box. In DOSSHELL.HLP, the S130 code is surrounded by exclamation marks. The second exclamation mark is immediately followed by an @ symbol, which indicates the start of the dialog box definition. The definition is terminated by the following code:

```
--@
```

Unlike help text defined in the Shell, help text in DOSSHELL.HLP does not use ^m symbols to create line breaks. Instead, all of the line for a paragraph is entered as a continuous string of text, and line breaks are created simply by pressing Enter←┘ and starting new lines. Browsing through DOSSHELL.HLP will yield numerous examples.

The first line of text in the DOSSHELL.HLP definition is interpreted as the title for the dialog box. Thus, the box that appears for the above text is titled, More on MS-DOS Editor. The rest of the text appears as the body of the dialog box.

To add linked help text dialog boxes to your own applications, simply mimic what is done for the Shell help files. First, when defining the initial help box in the Shell, define a selectable text string by enclosing it in double quotes and following it immediately with a code that will link to the DOSSHELL.HLP file.

You must make up your own codes, but each one must be unique. They can consist of any string of alphabetic and numeric characters. For consistency's sake, it is a good idea to follow the design used by the Shell — a single alphabetic character followed by a three digit number. It is also useful to choose an alphabetic character that the Shell does not use, so that it is easy to identify your modifications to DOSSHELL.HLP.

After you have added the appropriate help within the Shell, edit the DOSSHELL.HLP file. As with DOSSHELL.INI, it is essential that you use a word processor that produces plain ASCII files, and that handles lines longer than 256 characters. The Editor program that comes with the Shell is a good choice.

To add the linked text to DOSSHELL.HLP, go to the bottom of the file and enter the dialog box definition. This should take the following form, where XXXX represents the linking code:

```
!XXXX!@
Dialog Box Title
Text for dialog box.
--@
```

You may include additional linked dialog boxes by repeating this technique within the DOSSHELL.HLP dialog box definition: reference the next dialog box using tildes around the code, then create the box definition as described here. Two such linked boxes are defined in Figure 2-62.

```
!X100!@
Title For Linked Screen Number 1
This is the first help screen linked to the one in the Shell.
To see the next screen, select this: " Next Screen "~X101~
--@
!X101!@
Title For Linked Screen Number 2
This help screen is linked to the previous linked help screen,
not to the parent screen in the Shell.
--@
```

Figure 2-62 — Definitions for Two Linked Dialog Boxes in DOSSHELL.HLP.

Note: Keep in mind that when you delete a Shell program, the DOSSHELL.INI file is automatically updated. The program definition, including any online help, is deleted from it. But the Shell does not expect you to be adding linked dialog boxes in the DOSSHELL.HLP file, so that file is left intact. Consequently, if you edit DOSSHELL.HLP to add linked dialog boxes, then delete the program they belong to, you may want to go back and edit DOSSHELL.HLP to remove them. Their presence won't hurt anything, but in general, it is a good practice to remove outdated information.

If you do modify DOSSHELL.HLP, and want to move the Shell setup to another machine, you must copy DOSSHELL.HLP in addition to DOSSHELL.INI, to bring along the modifications to the Help system.

Advanced Color Adjustments

If you don't find a color scheme that you like among those that are provided with the Shell, you may modify the existing schemes, or create your own, by editing the DOSSHELL.INI file. The section that defines the color schemes begins on line 307 of Listing 2-1. Each color scheme has the following components: a title, a set of foreground colors for 17 different screen items, and a set of background colors for the same 17 screen items. The definition for the "Basic Blue" color scheme is on lines 309 through 352 of Listing 2-1. The 17 screen items for which foreground and background colors can be defined are shown in Table 2-2.

Table 2-2 — Screen Items for which Colors May Be Defined.

Item	Description
base	Defines the color of file names, directory names, program names, and lines for drawing the directory tree and borders around the various screen areas.
highlight	Defines the color of the MS-DOS Shell title, the currently selected area title, and the currently selected item in the ACTIVE area (see "selection" colors, following).
selection	Defines the color of the items that are currently selected in NON-ACTIVE areas — if the File List area is active, then the currently selected items in the Directory Tree, Program List, etc., will appear in this color, while the currently selected item in the File List will appear in the "highlight" color setup.
alert	Defines the color of text that appears in various alert boxes — attempting to read a disk drive that has no disk in it will display a warning box whose colors are defined by this setting.

continued...

...from previous page (Table 2-2).

Item	Description
menubar	The menubar option has no effect. Setting it to foreground black and background white is suitable.
menu	Defines the color of the bar menu and pulldown menus and the status bar at the bottom of the screen.
disabled	Defines the colors for menu options that are not currently available — if Directory Tree is the active area, then the Print option on the File menu is not available and will therefore appear in the colors defined for this item.
accelerator	Defines the colors for the for a menu item. This has no effect in graphics mode because in graphics mode the hotkey is simply underlined. In text mode its colors are set by this item. The hotkey is the key that jumps to a menu choice — on the File menu the letter D is the hotkey for the Delete option.
dialog	Defines the colors for dialog boxes. The background is the entire background of the dialog box, while the foreground is the color of the text.
button	Defines the colors for selection buttons in dialog boxes.
elevator	Defines the colors of the box in the scroll bar that indicates the section of a list that is currently displayed. Only the background has effect here, so normally foreground and background are set to the same color. If they are not set to the same color, the foreground is ignored.
titlebar	Defines the colors of the title bar for non-active areas — if File List is active, then this sets the colors for the title bar of the Directory Tree area, the Program List area, etc. The colors of the title bar for the active area are set by the highlight color (see above).

continued...

...from previous page (Table 2-2).

Item	Description
scrollbar	Defines the colors of the scrollbar other than the elevator. In graphics mode, the foreground setting has no effect. In text mode, the foreground is the color of the arrowheads at the top and bottom of the scrollbar.
borders	In graphics mode, defines the color of dialog box borders. Only the foreground has an effect, so the background is ignored — usually the two are set to the same color. Also in graphics mode, sets the border color of the pulldown menus. This has no effect in text mode.
drivebox	Defines the colors of the entire band near the top of the screen, in which the drive icons are contained. This does not affect the letters themselves in text mode, but the foreground sets the color of the icons in graphics mode. The foreground also sets the name of the path that appears in the box.
driveicon	Defines the colors of the letter in the drivebox area.
cursor	In text mode, defines the colors of the mouse cursor — the background is the cursor color, the foreground is the color of the number or letter that is highlighted by the cursor. This has no effect in graphics mode.

As mentioned above, each item is set twice — once for the foreground and once for the background. The foreground sets the color of the text that appears in an area, while the background is the color surrounding that text. For example, setting the foreground of the “base” setting to black and the background to green will display file names, program names, and directory names, in black letters on a green background.

The following 16 colors may be used to define the foreground and background of each item:

- black
- brightblack (this is really dark gray)
- blue
- brightblue
- green
- brightgreen
- cyan
- brightcyan
- red
- brightred
- magenta
- brightmagenta
- brown
- brightyellow
- white (this is really light gray)
- brightwhite (this is white)

Color names must be spelled exactly as shown above. To modify one of the existing color schemes, simply replace the current color with the color of your choice. If you want to create a completely new scheme of your own, the easiest thing to do is to copy one of the existing schemes, then edit the copy. The new copy should go immediately above the right-hand curly brace (}) that appears above the word "associations." This curly brace is what tells the Shell that it has read to the end of color scheme definitions, and that what follows is the Associations definitions.

If you use this technique to create a new color scheme, you must be sure to change the title to a unique name. In Listing 2-1, lines 655 through 698 contain a custom color scheme named "Rainbow."

When customizing colors in DOSSHELL.INI, you will have to experiment some — certain colors are not recognized for some of the items, and some foreground/background combinations are impossible to read. However, customizing the color can be fun, and enables you to design your own unique screens.

Chapter 3

The DOS Command Line

To make the computer do something, you must give it a command. On computers that use DOS 5.0, there are two ways of giving commands: by using the DOS 5.0 Shell and its menus; and by typing commands on the DOS command line. When the computer is booted it will display either the Shell screen or a command prompt, depending on how DOS 5.0 was configured during installation. Using the Shell was the topic of Chapter 2 — this chapter will discuss working at the command prompt.

The command prompt is a symbol that indicates the computer is waiting for a command. The appearance of the command prompt can be modified, but it is usually left the way DOS 5.0 sets it up during installation, which is to display the current drive and directory, followed by a greater-than sign (>), followed by the blinking cursor. For example, the following command prompt indicates that the current drive is C: and the current directory is the root directory:

```
C:\>_
```

A somewhat less common prompt is one that simply indicates the current drive, but does not indicate the directory:

```
C>_
```

Many other command prompts are possible — in fact, there is a DOS command that allows you to configure the prompt to be almost anything you like. The actual symbols used for the command prompt do not affect the behavior of the computer; whatever its appearance, it simply indicates that the computer is waiting for a command. Customizing the appearance of the prompt is discussed later in this chapter.

The "DOS command line" is the rest of the line following the prompt. This is the space where you type in the commands. The phrases "Enter a command on the command line" and "Enter a command at the DOS prompt" are both commonly used and are synonymous. They mean that you should type the command on the command line, and press Enter↵. When a command is typed at the command prompt, it appears on the screen but nothing is done with it until you press Enter↵, at which time DOS sends the entire command to the CPU.

One of the simplest commands is the one that tells DOS to change the current drive. This command consists of the new drive letter, followed by a colon. For example, if a computer has two disk drives, A and C, and the current drive is C, then typing **A:** and pressing Enter↵ will make A the current drive (the prompt will change from C:\> to A:\>).

There are two kinds of commands that can be entered on the command line: program commands, which run software programs that you have purchased or written yourself; and DOS commands. From the computer's point of view, there is no real distinction between these: DOS commands are simply programs that come with DOS. However, from a human perspective it is useful to distinguish between them because DOS commands are generally used for managing and maintaining computer resources (such as disks, files, and memory), while programs are used for accomplishing "real" work. For the purposes of this chapter, the terms "program name" and "command" are used interchangeably.

In order to use the command line effectively, it is necessary to have a basic understanding of how files are organized on disks. If you need more information, you can find a basic explanation of files and directories in Chapter 2, in the section titled, "Managing Files and Directories in the Shell." Chapter 5 provides a detailed discussion of this topic in the section titled, "Understanding and Managing Files and Directories."

When a command is entered on the command line, it instructs the computer to run a program. There are two places that programs might be stored: in the computer's memory, or on a disk. The only programs that are stored in memory are those for some of the most common DOS commands. These are contained in a file named COMMAND.COM, which is loaded into memory as part of the computer's boot sequence, and stays there until it is turned off. For example, the DIR command, which displays the files in a directory, is stored in memory. By storing these programs in memory, DOS can execute them very quickly. These are all small programs, so they don't occupy a lot of memory. Since they are used so frequently, it is worth giving up the memory they need to make them execute quickly. Furthermore, since they are in memory, they will always be available — they will not require a particular disk or directory to be available in order to run them (and they cannot be accidentally erased). DOS programs that are stored in memory are referred to as "internal" commands. Those that are stored on disk are called "external" commands.

Understanding the Path

In order for DOS to locate the file for an external command or program, it must know where to look. Hard disks can store thousands of files, so it could be quite time consuming for DOS to search through all of the file names on a disk every time you wanted to run a program. On the other hand, it is a nuisance to have to always type in the names of all the directories that lead to the program file every time you want to run it. To make this easier, DOS provides a command called

PATH. The PATH command allows you to generate a list of directories that DOS searches to locate programs. This enables you to run commonly used programs from anywhere in the system. PATH is an internal command, so it is stored in memory and is always available. For more information on using the PATH command, see Chapters 5 and 10.

When you enter a command, DOS first looks to see if it is one of the internal DOS commands stored in memory. If it isn't found there, DOS next looks in the current directory of the current drive. And if the program is not found there, DOS starts searching each directory in the Path. If the program still cannot be found, DOS displays the message "Bad command or file name." In this case, you must specify the full path name of the program. The term "full path name" refers to the name of a program file, prefixed with the complete drive and directory path leading to it. Let's take a program named WP.EXE located in the \WORDPROC directory, on the C: drive, for example. If the current directory is not C:\WORDPROC, you must enter C:\WORDPROC\WP to run the program (you can just enter \WORDPROC\WP if the current drive is C).

To see the current Path, enter the command PATH and press Enter←. The list of directories that DOS will search is displayed, with each directory separated from the previous one by a semi-colon. Here is a typical path for a computer with a single hard drive:

```
C:\;C:\DOS;C:\WP;C:\123;C:\RBFILES
```

This Path tells DOS to search the root directory (C:\), the C:\DOS directory, the C:\123 directory, and the C:\RBFILES directory, when it cannot locate a program file in the current directory. Note that for each directory, the letter of the drive that contains the directory is also included (in this example, the drive is always is C:). Floppy drives are not usually included on PATHs, for three reasons:

1. Access to floppy disk drives is very slow, so having DOS search the disk in a floppy drive is not efficient
2. Floppy disks are frequently removed from their drives; if DOS attempts to search for a drive that has no disk in it, it will display an error message and wait for you to put a disk in
3. Even if there is a disk in the floppy drive, there is no certainty that it will contain the directory and program that you want to run.

When DOS 5.0 is installed, it automatically puts a PATH command in the AUTOEXEC.BAT file. The AUTOEXEC.BAT file executes every time the computer is started, so the PATH command found there creates a Path that takes effect when the computer is booted. The contents of the Path command in AUTOEXEC.BAT will vary, depending on the configuration of your computer before installing DOS 5.0. The person who set up your computer may have further modified the Path.

Syntax of Commands

The syntax of commands is the same whether they are DOS commands or program commands. Commands may consist of more than just the command or program name; in fact, they may have up to four parts, in the following order:

Location Command Parameters Switches

Location: Identifies the specific drive and directory in which the program file is stored. The location is only required if the program is not in the current directory, or a directory specified in the Path. When a location is specified, it must be joined directly to the command name — there must be no space between them. For example, to run a program named WP on a disk in the A: drive, the following command would be entered:

```
A:\WP
```

Since the location identifier is joined directly to the command name, with no space separating them, it is sometimes considered part of the name. This is then referred to as the full path name for a command.

A location specifier is not required for internal DOS commands, because they are stored in memory. On computers with a hard disk, the DOS 5.0 install program puts the programs for external DOS commands on the hard disk, in a directory named \DOS, which is part of the Path. Consequently, unless the Path is modified to remove this directory, or these files are moved to a different directory, a location specifier is not required for DOS commands.

Command: The command name itself, which may be up to eight characters long, is the name of the command or program you want to run. File name extensions are not included when entering the command name — to run the program WP.EXE, simply type WP and press Enter←. The name is the only part of the command that is always required for all commands.

Parameters: Parameters indicate items that the command should operate on. For example, when starting a word processor you might enter `WP JIM.LET` where `JIM.LET` is a parameter that tells the WP program to load the file JIM.LET for editing. Parameters save time, since commands to load files into a program don't have to be entered after the program starts. Some commands take more than one parameter; an example is the DOS COPY command, which copies a file from one disk and/or directory to another. This command frequently takes two parameters: the first indicating the file to be copied, and the second indicating the location it should be copied to. For example, the following command copies the file named JIM.LET from the C:\WP\DATA directory to the A: drive:

```
COPY C:\WP\DATA\JIM.LET A:
```

COPY may actually take many parameters, since you can copy several files together into one large file. The details for the COPY command are discussed in Chapter 5.

For most commands, parameters are optional. However, some commands require them, and some don't use them at all. Parameters must be separated from the command name by a space.

Switches: Switches modify the way a program works, or how it interacts with the computer. Switches are usually entered as a forward slash (/), followed by the switch specifier. The switch specifier usually consists of a single letter or a number. Switches for most DOS commands are listed in Chapter 10.

Switches can have many different uses. Some examples are:

- to control how much memory a program uses
- to specify whether a program should use color or not
- to indicate whether a program should use a mouse
- to control how frequently a program should create backup files.

Not all programs or DOS commands use switches. Of those that do, each has its own unique set, so you must read the documentation for each program or command, to determine what switches it uses and what they mean. When more than one switch is specified, they may all be run together, as in /W/A/S, or they may be separated with a space, as in /W /A /S. Since the slash tells DOS that what follows is a switch, a space is not required between the slash and the command name or parameters that precede it.

An example of a DOS command that accepts switches is the DIR command, which displays the names of files in a directory. /P causes the DIR command to pause and wait for a key to be pressed whenever the screen is full. This prevents file and directory names from scrolling off the top of the screen before you have a chance to see what they are.

An example of a command that may use all four parts is the DOS EDIT command, which starts the DOS text editor. This is an external command. On a computer with a hard drive, the program file for EDIT is normally stored in the \DOS directory. If that directory has been removed from the Path, then starting EDIT requires specifying the location for the file. Assuming these conditions, here is a sample EDIT command:

```
C:\DOS\EDIT A:MYFILE.TXT /B
```

The location specifier in this command is C:\DOS; the command name is EDIT; the single parameter is A:MYFILE.TXT, which tells the editor to load a file named MYFILE.TXT from the A: drive; and the switch is /B, which tells EDIT to run in black and white mode.

Using the DOS Command Line

The DOS command line is limited to 127 characters. If you attempt to type more than 127 characters for a single command, the computer will stop displaying characters after the 127th and beep whenever a key is pressed. Since the computer screen is only 80 characters wide, long commands will wrap to a second line. This happens automatically — Enter← cannot be used to move to the next line because as soon as Enter← is pressed, DOS assumes the full command has been entered and executes it. Just keep typing when you reach the end of the line, DOS will jump down to the beginning of the next line and continue accepting your keystrokes. It doesn't matter if the break occurs in the middle of a word, it's all the same to DOS.

While 127 characters seems like a lot, complex programs sometimes involve a large number of parameters and switches. If data files specified in the parameter list are not in the current directory, then each one must have its full path name included in the parameter list. Consequently, 127 characters is not always sufficient for program commands. The SUBST command can be used to create abbreviations for long paths. SUBST is discussed in detail in Chapter 6.

DOS ignores leading blanks on the command line, so if you happen to hit the space bar before typing a command it doesn't matter. However, leading blanks do count as part of the 127 maximum. When running macros created with the Doskey program, however, the macro name must begin in the first position after the prompt. Otherwise, the macro will not be executed. This distinction between how leading blanks are handled for macros and commands can be exploited to create macros that have the same name as a command, without disabling the command. Macros are discussed in detail in the "Using Doskey To Create Macros" section later in this chapter.

←BkSp is used to make corrections to the command line. Each time ←BkSp is pressed, the cursor is moved back one space and the character that occupied that space is erased. There is a limitation to this, though: if a command wraps to a second line, ←BkSp will not move the cursor back up to the first line. Consequently, when two-line commands are entered, the first line should be checked and corrected, if necessary, before the cursor moves to the second line. ← functions in the same way as ←BkSp.

Shortcuts for Re-entering and Editing Commands

←BkSp is of limited use, particularly when you need to make a number of corrections to a command, or when corrections are needed on the first line after the second line has been started. Furthermore, it is often useful to be able to repeat a command, with some minor modifications. DOS 5.0 has several features that make it easy to re-enter and edit commands that have already been typed.

When a command is entered, DOS stores it in an area of memory that has been reserved for just this purpose. This area is called the "template." Keystrokes stored there can be played back and edited. The template is 127 bytes long, and always contains the most recently entered command. Whenever a new command is entered, the previous command is overwritten by the new one. A command is not stored in the template until Enter↵ is pressed. For example, typing the letters AAA, then backspacing over them, will not put them into the template.

There are several keys on the keyboard which provide access to the template. All of these keys operate only on the template, and therefore only on the most recently entered command. As a result, it is not possible to use these keys to retrieve commands that were entered prior to the one in the template. These keys are discussed in detail in the following paragraphs, and are summarized in Table 3-1 at the end of this section. The DOSKEY program, discussed later in this chapter, allows you to create a "command stack" that remembers multiple commands as they are entered, and lets you recall them for replaying or editing.

There are three ways in which the template can be played back. Note that playing back the template only displays the stored characters on the screen, it does not execute the command. Enter↵ must be pressed before the command can be carried out.

The first, and simplest, means of replaying the template is the use of F1. Each time F1 is pressed, it plays back the next character in the template. The keyboard's typematic feature works with F1, so holding the key down will play back the characters in the template, until the end is reached or the key is released. This is convenient when you want to repeat only the beginning part of a command, either to use the first part by itself or to give it a different ending. For example, suppose you want to copy three files, named JIM.LET, FINANCE.DOC, and ACCOUNTS.WKS, from the C:\WP\DATA directory to the A: drive. The following command copies the first file:

```
COPY C:\WP\DATA\JIM.LET A:
```

Now, rather than retyping the first part of the command, you can press F1 or → 16 times to repeat the first part of the command:

```
COPY C:\WP\DATA\
```

The command can be finished by typing in the next file:

```
FINANCE.DOC A:.
```

You may insert something into the middle of a command by playing back the first part of it, then pressing Ins and typing the characters to be inserted, then using F1 to finish the command. For example, suppose you intended to perform the above COPY command, but you left out the \DATA subdirectory by mistake. The template would contain this command:

```
COPY C:\WP\JIM.LET
```

When DOS attempts to execute this command, it will not be able to find the file and will display the following message:

```
File not found - C:\WP\JIM.LET
        0 file(s) copied
```

To fix the problem without retyping the entire command, use F1 to replace the first part of the command:

```
COPY C:\WP\
```

Press Ins, type the missing information `DATA\` then use F1 to finish the command. Insert mode can then be turned off by pressing Ins a second time.

In a similar way, characters can be deleted from the middle of a command. To accomplish this, press F1 to replay the command up to the part to be deleted, then delete is pressed, once for each character to be deleted. Finally, F1 is used to replay the rest of the command.

The second technique for playing back the template is to use F2. Pressing F2, followed by a single character, plays back the characters in the template, up to but not including the first occurrence of the character typed. Again using the above example, pressing F2, then D, would put the following command on the screen:

```
COPY C:\WP\
```

Now the command may be finished in whatever way you like.

F2 may be used repeatedly to get to the second, third, or later occurrence of a particular key. For example, if the command `DIR C:\WP\DATA\LETS` were entered, the files in the LETS directory would be listed. Now, to list the files in the C:\WP directory, press F2, then \, then F2 and \ again. This would display the command:

```
DIR C:\WP
```

The third technique for playing back the template is to use F3. When you start to type in a new command, you overwrite the template, one character at a time. F3 plays back the template from the current cursor position. For example, suppose you wanted to copy the file C:\WP\DATA\JIM.LET to the A: drive, then wanted to use the COMP command to make sure the copy was successful. After the COPY command, the template looks like this:

```
COPY C:\WP\DATA\JIM.LET A:
```

To execute the COMP command, all you have to do is type the word COMP, then press F3. The following command would appear on your screen:

```
COMP C:\WP\DATA\JIM.LET A:
```

If all you want to change in a command is the very end of it, F3 can be pressed at the command prompt to replay the entire command, then use ←BkSp to delete the end. Also, if you have used F1 and ins to insert something into a command, using F3 to play back the rest of the command is faster than holding down F1 while it plays back each character individually.

There are four more keys used for affecting the template, all though they do not play it back directly. These are F4, F5, F6, and esc.

F4, like F2, is used in conjunction with a character key. However, where F2 plays back everything up to but not including the character, F4 deletes everything up to but not including the character, starting with the current position. For example, if you play back the first three characters using F1, then press F4, then the character that appears in the sixth position, the fourth and fifth characters are deleted from the template. Pressing F3 after F4 plays back what is left. Alternatively, you may use ins to insert a new beginning for the command before pressing F3, as in the following example.

If the template contains the command, COPY C:\WP\DATA\JIM.LET A:, then pressing F4, then J, then F3, will display the following:

```
JIM.LET A:
```

If, after pressing F4 and J, Ins was pressed, followed by the characters, COPY A:, then press F3, the following would appear on the screen:

```
COPY A:JIM.LET A:
```

Now, the final A: could be changed to whatever drive the file should be copied to, using ←BkSp, and the new command would be ready.

As mentioned above, a new command is only stored in the template after Enter↵ is pressed to execute the command. This is usually acceptable, but there are times when it is desirable to store a command in the template without executing it. For example: if a command has extended to a second line before an error is spotted on the first, you don't want to attempt to execute the command before the error is corrected. However, there is no direct way to get back to the first line.

F5 can be used to store a command in the template without executing it. F1 can then be used to replay and correct the first part of the command, and F3 used to replay the rest. In fact, F5 is useful for any long command with an error near the beginning, even if the entire command fits on one line. It is quicker to press F5 to store the command, use F1 to correct the error, then F3 to replay the rest, than it is to backspace over the entire command, fix the error, then retype it.

F6 can be used to place a control-Z symbol (^Z) in the command line. **^Z** is a special symbol that means "end of file." It is used to indicate to DOS, or to any other program, that the end of the file has been reached. Even though it occupies two spaces on the screen, **^Z** is really just one symbol. Pressing F6, or holding down Ctrl and pressing Z, will create a **^Z** symbol. It is not common for F6 to be used on command lines, but a few programs expect a **^Z** symbol to appear as part of a parameter.

esc is used to cancel a command, without executing it and without affecting the template. Pressing esc moves the cursor to the next line, but no command prompt appears there. Nonetheless, you may go ahead and enter a new command at that point.

Table 3-1 summarizes the keys that may be used for editing the command line.

Table 3-1 — Keys Used for Replaying and Editing Commands.

Key	Effect
F1 or →	Plays back one character at a time from the template.
F2 + Character	Plays back the template, up to but not including the first instance of Character.
F3	Plays back the template, from the current position to the end.
F4 + Character	Deletes characters from the template, starting at the current position and continuing up to but not including Character.
F5	Copies the current command line to the template, but then terminates the command without executing it, as though esc had been pressed.
F6	Places a control-Z (^Z) character into the command line.
←BkSp or ←	Deletes the character to the left of the cursor on the command line, without affecting the template.
delete	Deletes the character in the template that appears in the current position.
ins	Turns on Insert mode, causing characters typed to be inserted into the template at the current position. When Insert mode is on, pressing Insert again turns Insert mode off.
esc	Cancels the current command line, without affecting the template.

These keys can be used in many ways, and in many combinations, to provide very efficient control over the replaying of commands. For example, the command

```
COPY C:\WP\DATA\JIM.LET A:
```

can be converted to

```
COPY A:\DATA\JIM.LET C:
```

with the following steps:

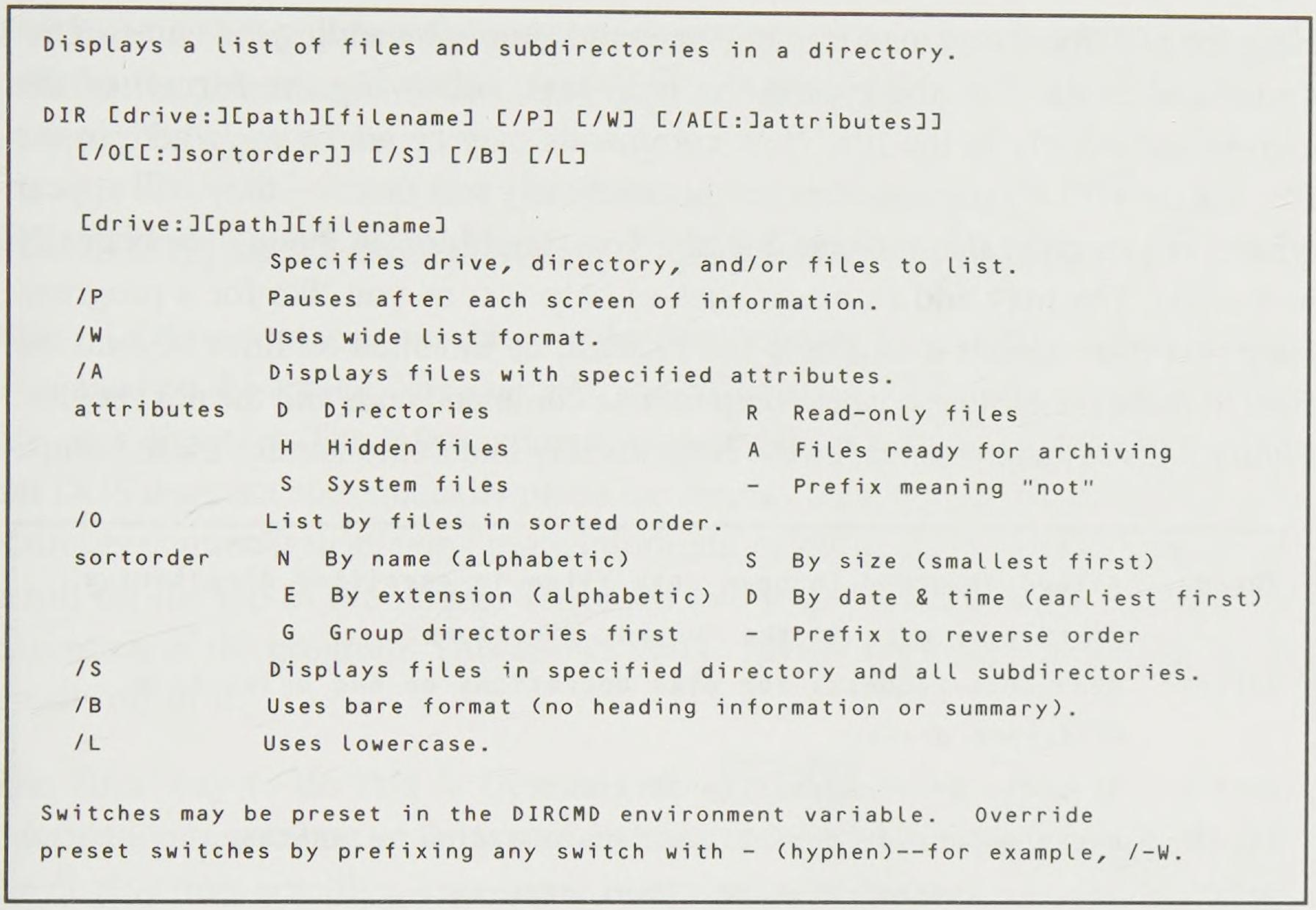
1. Press F2, then C, which brings up COPY.
2. Press F4, then D, to delete C:\WP\ .
3. Press Ins and type A:\ , to insert the new drive identifier.
4. Press F2, then A — the command line now contains COPY A:JIM.LET.
5. Type **C:** and press Enter← to execute the command

In these steps, the 23 keystrokes that would have been required to type the new command from scratch have been replaced with 12 keystrokes. If you work at the command prompt a lot, it is worth becoming proficient with the command line editing keys. You should also learn to use the DOSKEY program, discussed later in this chapter.

Obtaining Online Help for DOS Commands

DOS 5.0 provides online help for all of the DOS commands. There are three ways to obtain help from the command line:

1. You may obtain a brief summary of all DOS commands by typing the word HELP and pressing Enter←. This simply displays a file named DOSHELP.HLP, which contains a one or two line summary of each DOS command.
2. You may obtain more extensive help by typing the name of the command followed by the question mark switch (/ ?). For example, typing `DIR /?` will display the screen shown in Figure 3-1.



```
Displays a list of files and subdirectories in a directory.

DIR [drive:][path][filename] [/P] [/W] [/A[[:]attributes]]
  [/O[[:]sortorder]] [/S] [/B] [/L]

  [drive:][path][filename]
              Specifies drive, directory, and/or files to list.
  /P          Pauses after each screen of information.
  /W          Uses wide list format.
  /A          Displays files with specified attributes.
  attributes   D  Directories                R  Read-only files
               H  Hidden files               A  Files ready for archiving
               S  System files               -  Prefix meaning "not"
  /O          List by files in sorted order.
  sortorder    N  By name (alphabetic)       S  By size (smallest first)
               E  By extension (alphabetic)  D  By date & time (earliest first)
               G  Group directories first    -  Prefix to reverse order
  /S          Displays files in specified directory and all subdirectories.
  /B          Uses bare format (no heading information or summary).
  /L          Uses lowercase.

Switches may be preset in the DIRCMD environment variable.  Override
preset switches by prefixing any switch with - (hyphen)--for example, /-W.
```

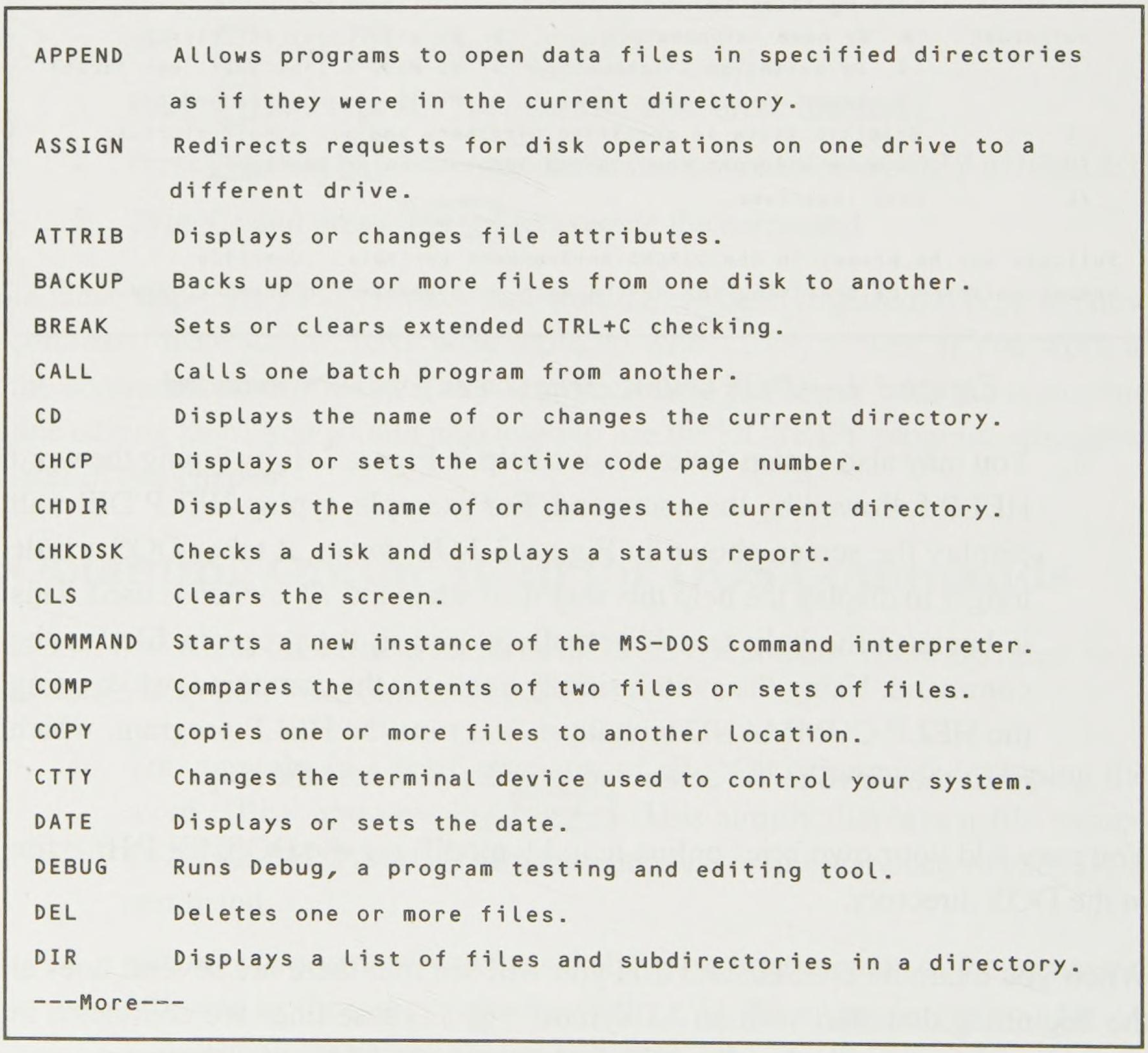
Figure 3-1 — DOS Online Help Screen for DIR Command.

3. You may also obtain the extensive help in Figure 3-1, by typing the word HELP followed by the command. For example, typing HELP DIR will display the screen shown in Figure 3-1. However, it takes DOS a little longer to display the help this way than when the **/?** switch is used. This is because the help text is actually stored in the program file for the command. Using the switch simply executes the command, while using the HELP COMMAND technique first runs the HELP program, which must then invoke the command in order to access the help.

You may add your own brief online help by modifying the DOSHELP.HLP file in the DOS directory.

When you examine DOSHELP.HLP, you will see that there are several lines at the beginning that start with an AT symbol (@). These lines are comments in the file, and are not displayed by the Help program. Additional comments may be added by starting a line with **@** and typing the comment.

Help for additional commands may be entered simply by adding the name of the command to the file and typing the help text, following the format of the commands already in the file. New commands may be added anywhere in the file, but the HELP program does not automatically sort them — they will appear wherever you enter them. Consequently, you should locate them alphabetically in the file. You may add as many lines of help text as you like for a program. Help text that extends to multiple lines should be indented on lines beyond the first, to make it easy to see where help for one command ends and the next begins. Figure 3-2 shows one screen of the Help display that demonstrates these points.



```
APPEND   Allows programs to open data files in specified directories
         as if they were in the current directory.
ASSIGN   Redirects requests for disk operations on one drive to a
         different drive.
ATTRIB   Displays or changes file attributes.
BACKUP   Backs up one or more files from one disk to another.
BREAK    Sets or clears extended CTRL+C checking.
CALL     Calls one batch program from another.
CD       Displays the name of or changes the current directory.
CHCP     Displays or sets the active code page number.
CHDIR    Displays the name of or changes the current directory.
CHKDSK   Checks a disk and displays a status report.
CLS      Clears the screen.
COMMAND  Starts a new instance of the MS-DOS command interpreter.
COMP     Compares the contents of two files or sets of files.
COPY     Copies one or more files to another location.
CTTY     Changes the terminal device used to control your system.
DATE     Displays or sets the date.
DEBUG    Runs Debug, a program testing and editing tool.
DEL      Deletes one or more files.
DIR      Displays a list of files and subdirectories in a directory.
---More---
```

Figure 3-2 —A Segment of the DOSHELP.HLP File.

The line "--- More ---" at the bottom of the screen, indicates that there is more information than can fit on one screen. Pressing any key will display the next screen of information.

Pausing and Stopping DOS Commands

Many DOS commands produce some form of output — information that is displayed on the screen. For example, the DIR command displays a list of all the files in a directory. The information displayed does not always fit on one screen, but DOS does not automatically pause the display after the screen is full. Instead, it simply continues displaying more information, letting the previous information scroll off the top of the screen. There are two ways to temporarily suspend the execution of the program. This allows you to review the screen before anything scrolls off of it.

The first way to do this is by using the Ctrl-S key combination. When information is scrolling up the screen, holding down Ctrl and pressing S will stop the display from scrolling. Pressing any key on the keyboard (except ⇧Shift, Ctrl, alt, or pause) will restart the display. This can be repeated as often as you like for any command.

A somewhat more efficient way of controlling the display is to use the MORE command along with the Pipe symbol (|). The Pipe symbol is discussed in detail in the next section of this chapter. To use the MORE command, simply follow any command with a space, followed by the pipe symbol, another space, and then the word MORE. (The spaces are optional, but they make the command more readable.) The output from the command will now appear as usual, but when the screen is full the display will stop scrolling and display the line, "--- MORE ---" at the bottom of the screen. Pressing any key (except ⇧Shift, Ctrl, alt, or pause) will display the next screen.

The MORE command is useful when you know you want to pause the display after each screen. The Ctrl-S (Pause) option is more useful if there are many screens, and you only want to pause the display once or twice while they scroll by. MORE can be used with any program that displays information directly onto the DOS screen. Programs that take control of the screen, like word processors, cannot be used with MORE.

DOS commands may be stopped in the midst of execution by holding down Ctrl and pressing either C or break. Other programs may or may not respond to these keystrokes, depending on how they were written. Occasionally, a program will not respond to Ctrl-C, but will respond to Ctrl-break. On the other hand, all programs that respond to Ctrl-C will also respond to Ctrl-break, so it is a good idea to rely on Ctrl-break for this purpose.

You may want to terminate a command that you no longer want to execute, but that will take awhile to complete. For example, you may start a directory listing of a very long directory, then decide you don't want to see it after all. Ctrl-break will end it for you. However, if part of a command has already been carried out, that part will not be undone by Ctrl-break. For example, if the command DEL *.* was entered, instructing DOS to delete all files in the current directory, then pressing Ctrl-break would only save the files that had not been deleted, any files that were deleted before you pressed them would be gone.

Redirecting Command Line Input and Output

DOS commands receive their input from the keyboard, and send their output to the screen. This is what is usually wanted, but sometimes it is useful to be able to use some other source for the input, and to send the output to some destination other than the screen. For example, it might be useful to store a DIR listing in a file.

Changing the source of input, or destination of output, for a command is known as "redirection," and is accomplished with the greater-than (>) and less-than (<) symbols. The greater-than symbol is used to redirect output, while the less-than symbol is used to redirect input. These can be remembered by thinking of them as arrowheads that point to the object they are redirecting. The `<` symbol points to the left, which is where input comes from, and the `>` symbol points to the right, which is where output goes. To store a listing of the current directory in a file named LIST.DIR, the following command could be used:

```
DIR > LIST.DIR
```

STOP

If this command is executed when a file named LIST.DIR already exists, the existing file will be overwritten with the new one. Be careful with this command —

there is no safety net. If you would prefer to append the new listing to the old one, use two greater-than symbols (>>). The above command would then appear as:

```
DIR >> LIST.DIR
```

If LIST.DIR does not exist, it is created by `>>`, just as it would have been with a `>`. But if the file already exists, the output will be appended to the end of the existing file.

When there is an error in the way a DOS command is entered, DOS displays an error message. Some of these messages are the result of the command itself diagnosing the error, while other messages are the result of DOS diagnosing the error. Error messages diagnosed by the command are affected by the output redirection symbol, the others are not. For example, the command `DOSKEY XXX` produces the message "Invalid macro definition." This error message is produced by DOSKEY, so it can be redirected. If the command were entered as `DOSKEY XXX > ERROR.FIL`, the message would be stored in the file ERROR.FIL instead of appearing on the screen.

On the other hand, when the command `COPY XXX YYY` is entered, if there is no file named XXX, the following messages appear on the screen:

```
File not found - XXX
        0 file(s) copied
```

The first message, "File not found - XXX," is produced by DOS, not the COPY program, so it cannot be redirected. The second message, "0 file(s) copied," is produced by the COPY program, so it can be redirected. Consequently, when the command

```
COPY XXX YYY > ERROR.FIL
```

is executed, the first message (File not found - XXX) is displayed on the screen, while the second message (0 file(s) copied) is not, it is written to the ERROR.FIL file instead.

Output redirection is useful when you want the output of a command to be written to some destination other than the screen. For example, it can be used to send output to a printer. In most cases, your printer is connected to a port on the computer which is referred to as either LPT1 or PRT1 (the two names are synonymous). Consequently, entering the command `DIR>LPT1` will print a directory listing. If the LPT1 port is used for some other device, or if you have two printers, the LPT2 (or PRT2) port may be used.

You can also redirect the output of a command to be used as the input to one of the filter commands. In much the same way, input redirection can specify that a disk file be used as input to a command. However, you cannot use the `<` or `>` redirection symbols to specify that the output of one command is to be used as the input to another. To connect input and output using these requires two command sequences and an intermediate file. For example, to apply the MORE filter to the results of the DIR command, the following two commands could be used:

```
DIR > TEMP.FIL
MORE < TEMP.FIL
```

The Pipe symbol (|) can be used to simplify this process. It is called a Pipe because of the way it connects the two commands, sending the output of the first command into the input of the second. Using Pipe, the two commands listed above can be replaced with the single command:

```
DIR | MORE
```

The Pipe not only reduces two commands to one, but also eliminates the need for an intermediate file. Note that the blank spaces before and after the Pipe symbol are not required, but they make the command more readable.

The DOS Filter Commands — MORE, FIND, and SORT

Filter commands are used to divide, rearrange, or extract parts of their input. They are external commands. There are three filter commands:

- **MORE** accepts input, then displays it on the screen, pausing after each screen until a key is pressed.
- **FIND** is used to search through its input to locate a particular text string.
- **SORT** sorts its input alphabetically.

By using filters in conjunction with the redirection symbols, the input can come from a file, and the output can go to a file, a printer, or another command. The filter commands and their switches are summarized in the following table.

Table 3-2 — The DOS Filter Commands and their Switch Settings.

Filter	Switch	Effect
MORE	None	Displays the input on the screen, pausing for a key to be pressed after each screen.
FIND	None	Searches for a specified string in a specified file. Displays all lines in which the string appears.
	/ C	Produces a count of the lines in which the search string occurs. The lines themselves are not displayed.
	/ I	Ignores the case of the characters in the search string.
	/ N	Displays line numbers at the beginning of each displayed line.
	/ V	Displays all lines in which the search string does not appear
SORT	None	Reads its input, sorts it in alphabetic order, and sends the sorted data to the output.
	/+N	Bases the sort on column N of each line.
	/R	Reverses the sort order.

Filter commands are best used with plain ASCII files — those that do not contain any special formatting characters. If you see non-alphanumeric symbols on the screen when a file is displayed with one of the filters, then the file contains non-ASCII symbols. Most word processor, spreadsheet and database files contain non-ASCII formatting symbols. This makes them difficult, if not impossible, to interpret. The filters cannot be used with executable files — those with .COM or .EXE extensions — because they consist of computer instructions, rather than data that is stored in English.

MORE allows you to view the contents of a file, one screen at a time. It does this by redirecting the input (the file), through the filter (MORE), to the screen (the output). MORE allows data to scroll onto the screen, until the screen is full. It then adds the message "--- MORE ---" at the bottom of the screen and waits for an input from the keyboard. Pressing any key causes the next screen to be displayed.

To examine a file named MYFILE.TXT, enter the command `MORE < MYFILE.TXT`. If the file to be displayed is not in the current directory, a complete pathname must be specified:

```
MORE C:\WP\LETTERS\MYFILE.TXT.
```

FIND is used to determine whether an input source contains a particular string of characters. For the following examples, the FIND filter will be applied to a file named GREEN.LET, shown in Figure 3-3.

```
June 1, 1991


Mr. William Green, President
Green Industries, Incorporated
1900 Industrial Boulevard
Minneapolis, Minnesota 55411

Dear Mr. Green,

I am writing to thank you for having the sample products sent
to me last week.  My staff has begun reviewing and testing the
products, and should have recommendations on some of them by the
end of next week.  Our impression so far has been quite
favorable, and I am sure we will be able to work out a mutually
satisfying relationship.

Please let us know when your new product line is available, as
there are several items we are interested in.

Sincerely,


Susan Anderson
Vice President, Automation

P.S. The "Ultra" products are especially exciting.
```

Figure 3-3 — Sample Input File for the FIND Filter.

The following command searches the file in Figure 3-3 for the word "Green":

```
    FIND "Green" GREEN.LET
```

The result of this command is:

```
---------- GREEN.LET
Mr. William Green, President
Green Industries, Incorporated
Dear Mr. Green,
```

When using the FIND filter, the character string to be searched for is required, and must be enclosed in double quotes. The file name to search is not required, but if you do not enter one, FIND will display a blank command line and accept whatever you type as its input. To break out of the FIND command under these circumstances, press F6 followed by Enter←. F6 inserts a **^Z** character, which is the symbol used to indicate the end of a file. When FIND sees the **^Z** it assumes the input has ended and returns you to the command prompt.

If you want to search for a string that contains double quotes ("), you must enter the quotes twice in the search string. The following command successfully searches the sample file for the string, The "Ultra" products:

```
    FIND "The ""Ultra"" products" GREEN.LET
```

If the file to be searched is not in the current directory, its full path must be specified. The DOS wild cards (* and ?) cannot be used with FIND to search multiple files. However, FIND can be used in conjunction with the FOR command to search multiple files. FOR is discussed in detail in its own section, later in this chapter.

FIND does not recognize strings broken by carriage returns — symbols that indicate where one line ends and the next begins. Most text files include carriage return symbols at the end of each line, but some do not, in which case they consist of one or more long stream(s) of words. FIND will not locate a string that is broken by a carriage return. For example, The first full line of the letter shown in Figure 3-3 ends with the word "sent." The second line begins with the word "to." It is not possible to use the FIND filter to locate the string "sent to" in this file.

The output of the FIND command can be piped to the input of the MORE command, to prevent the FIND output from scrolling off the screen. For example, to search a file named SALES.MEM for the string "Midwest sales percentages," and prevent the output from scrolling off the screen, the following command could be used:

```
FIND "Midwest sales percentages" SALES.MEM | MORE
```

There are four switches that can be used with FIND: /C, /I, /N, and /V. The /C switch is used to display just a count of the lines that contain the specified string. The command FIND "Green" GREEN.LET /C displays the following line:

```
---------- TEMP.ASC: 3
```

By default, the FIND filter is case sensitive; using the file in Figure 3-3, the command FIND "green" GREEN.LET, would not locate any lines, because the search string was not capitalized. The /I switch is used to indicate that the search should be insensitive to case. The command

```
FIND "green" GREEN.LET/I
```

would locate the appropriate lines.

Simply displaying the lines that contain the search string is not always sufficient, particularly in a large file. It is important not only to see the line, but to know where it is in the file. The /N switch precedes each displayed line with its line number in the file. The command

```
FIND "Green" GREEN.LET /N
```

produces the following output:

```
---------- GREEN.LET
[5]Mr. William Green, President
[6]Green Industries, Incorporated
[10]Dear Mr. Green,
```

FIND normally displays all of the lines in the input that contain the specified string. Sometimes the opposite effect is desired — you want to see all lines that do not contain the search string. The /V switch is used to accomplish this. Using the file in Figure 3-3, the command

```
    FIND "Green" GREEN.LET /V
```

produces this output:

```
---------- GREEN.LET
June 1, 1991
1900 Industrial Boulevard
Minneapolis, Minnesota 55411
I am writing to thank you for having all the sample
products sent to me last week. My staff has begun
reviewing and testing the products, and should have
recommendations on some of them by the end of next week.
Our impression so far has been quite favorable, and I am
sure we will be able to work out a mutually satisfying
relationship.
Please let us know when your new product line is
available, as there are several items we are interested
in.
Sincerely,
Susan Anderson
Vice President, Automation
P.S. The "Ultra" products are especially exciting.
```

The /C switch can be used in combination with two of the other three switches. (When /C and /N are used together, the /N switch is ignored.) When /C and /I are used together, the result is a count of the lines that contain the search string, without case sensitivity. When /C and /V are used together, the result is a count of all the lines that do not contain the search string.

SORT is used to sort an input source, usually a file. The output can be the screen, a new file, or a device, such as a printer. SORT can handle files up to 64K bytes in size. The examples here will be based on a file named ADDRESS.LST, shown in Table 3-3.

Table 3-3 — Sample File for Use with the Sort Command.

```
Williams, Frank     1011 Zenith Ave. So.  Minneapolis, MN     55401
Andrews, Bill       14 North Snelling     St. Paul, MN        55408
Banks, Susan        2022 Hacienda Blvd.   San Diego, CA       91020
Yeoman, Mary        134 North Summit      Silver Spring, MD   32976
Hoffman, Andy       132 E. Franklin       Seattle, WA         80971
Manchester, Chris   9501 Drew Ave. So.    Bloomington, MN     55321
```

Unless the /+N switch is used, the SORT filter determines the sorted order based on the character in the first column of each line. If the first character of two lines match, it sorts on the second character, and so on. When considering what criteria SORT will use to determine the new order for a file, it is usually sufficient to expect it to sort according to the ASCII table, without distinguishing between upper and lower case letters — lower case letters are treated as upper case. A complete ASCII table appears in Appendix A. Since the blank space symbol appears before numbers and letters in the ASCII table, all the blank lines come out first.

Note: For the technically minded, SORT actually uses the collating sequence table that corresponds to the country code and code page setting for which your computer is configured. For characters that have ASCII codes greater than 127, SORT uses information supplied in the COUNTRY.SYS file, which is in the DOS directory, or in an alternate file that you may have specified using the COUNTRY command in your CONFIG.SYS file.

The input redirection symbol must be used to sort a file; otherwise, SORT operates on keyboard input until F6 is pressed, followed by Enter←. The following command sorts the file in Table 3-3 by the first character in each line:

```
SORT < ADDRESS.LST
```

The results of this command appear on the screen, as follows:

```
Andrews, Bill       14 North Snelling     St. Paul, MN        55408
Banks, Susan        2022 Hacienda Blvd.   San Diego, CA       91020
Hoffman, Andy       132 E. Franklin       Seattle, WA         80971
Manchester, Chris   9501 Drew Ave. So.    Bloomington, MN     55321
Williams, Frank     1011 Zenith Ave. So.  Minneapolis, MN     55401
Yeoman, Mary        134 North Summit      Silver Spring, MD   32976
```

There are two switches that can be used to modify the way SORT works: /R and /+N.

/R is used to reverse the sort order. In the above example, the command

```
SORT < ADDRESS.LST /R
```

would put the Mary Yeoman line at the beginning of the list, and the Bill Andrews line at the end.

/+N enables you to sort your data based on the characters occurring in a column other than the first column of each line (where N represents the column in which to start the sort). For example, to sort the ADDRESS.LST file in Table 3-3 by zipcode, which begins in column 60, and save the output to a file named ADDRSORT.ZIP, the following command would be used:

```
SORT < ADDRESS.LST > ADDRSORT.ZIP /+60
```

If sort decisions need to be made beyond the end of a data line, SORT will wrap back to the beginning of the line.

If you want to see the output of a SORT command on the screen, and there will be more than one screen of information, the output may be piped through the MORE filter. The following command sorts a file named NAMES.FIL based on the characters in position 10 of each line, and uses the MORE command to pause the display after each screen of information:

```
SORT < NAMES.FIL /+10 | MORE
```

Frequently Used DOS Commands

This section provides an introduction to some of the most frequently used DOS commands. This introduction is sufficient for most uses of the commands, but Chapter 10 provides detailed syntax diagrams and full explanations of how the commands work, and what parameters and switches they accept. Many of these commands are also discussed in the appropriate sections of this book. For example, the DEL, DIR, RENAME, and TYPE commands are used for file management, so they are covered thoroughly in Chapter 5. In addition to the commands discussed here, FIND, HELP, MORE, and PROMPT are frequently used commands that are discussed in detail in other sections of this chapter. The commands presented here appear in alphabetical order.

CLS

The CLS command clears the screen, and positions the cursor at the top left corner of the screen. CLS is useful for eliminating clutter from the screen, and for providing a clear screen for messages and command output during batch file execution.

DEL

The DEL command is used to delete one or more files from a directory. The simplest version of the command is:

```
DEL filename.ext
```

This deletes the specified file from the current directory. Paths may be specified to delete files in other directories, and DOS wild cards may be used to delete sets of files.

DIR

The DIR command is used to display a list of the files in a directory. Entering the command by itself, with no switches or parameters, displays all files in the current directory, except those whose Hidden attribute is set to ON. File names are displayed in all upper case letters. The size of the file, as well as the date and time of its last modification, are shown.

If you want to see information about just one file, then a file name may be specified, as in:

```
DIR JIMSMITH.LET
```

To see a group of files whose names match a pattern, the DOS wild cards (* and ?) may be used. The following command displays all of the files with a BAT extension:

```
DIR *.BAT
```

You may also specify a different directory than the current one. The following command specifies all files in the C:\WP\LETTER directory:

```
DIR C:\WP\LETTER
```

Switches can be used with the DIR command to cause the output to

- pause after each screen is displayed
- display in several columns across the screen, instead of one long column down the left side of the screen
- show only files whose attributes are set in a certain manner
- sort the file names according to any of several criteria
- display files from sub-directories
- omit the size, date, and time information from the display
- display in lowercase letters rather than uppercase.

All of these options are discussed in detail in Chapter 5.

DOSKEY

The DOSKEY command is used to load the Doskey program into memory, and to use the program after it is loaded. Doskey is used to recall DOS commands that have been entered, to enhance the command line editing capabilities, and to create macros. Two sections of this chapter, "Using the Doskey Command Stack" and "Creating and Using Macros with Doskey," are devoted to detailed discussions of the DOSKEY command.

FOR

The FOR command is used to repeat a DOS command several times, using a different file or text string from a specified list of files or text strings each time. For example, FOR can be used to execute the FIND command for all the files in a set of files; or to use the TYPE command (discussed below) to display each of a set of files, one at a time. FOR is used most commonly in batch files, but it can also be useful at the command prompt.

The FOR command has the following syntax:

FOR %variable IN (value list) DO command

Where **%variable** is a symbol that is referenced in the command parameters of the command that follows the word **DO**. The FOR command substitutes %variable for each value in the **value list**. The value list may contain a list of file names or text strings. For example, to use the FIND command to search all files with a BAT extension for the string "DOS," enter the following command:

```
FOR %A IN (*.BAT) DO FIND "DOS" %A
```

In this example, the value list consists of all file names in the current directory that have a BAT extension. Suppose the current directory has two such files: FILE1.BAT and FILE2.BAT. The FOR command will retrieve each of these file names, one at a time, and for each one execute the FIND command. Thus, the following two commands will be executed:

```
FIND "DOS" FILE1.BAT
FIND "DOS" FILE2.BAT
```

Using the Doskey program, the FOR and FIND commands can be combined into a macro that makes it much easier to search multiple files for a text string. The macro would be defined as follows:

```
DOSKEY FFIND=FOR %A IN ($2) DO FIND $1 %A
```

After this macro is defined, the FFIND command can be used to search a set of specified files for a specified text string. For example, to search all files with a BAT extension for the string "DOS," the following command would be entered:

```
FFIND "DOS" *.BAT
```

If you are not familiar with the DOSKEY command and macros, you will need to read those sections of this chapter to understand how the FFIND macro works.

Another use of the FOR command is to search a particular file for multiple text strings. The following command illustrates this:

```
FOR %A IN ("DOS" "TIME" "DATE") DO FIND %A AUTOEXEC.BAT
```

This command is equivalent to entering the following three commands:

```
FIND "DOS" AUTOEXEC.BAT
FIND "TIME" AUTOEXEC.BAT
FIND "DATE" AUTOEXEC.BAT
```

You may not use more than one FOR command on one command line. It is therefore not possible to search multiple files for multiple text strings in one command.

FOR may be used with the TYPE command to display a number of files on the screen. To display all batch files in the current directory, enter the following command:

```
FOR %A IN (*.BAT) TYPE %A
```

The output of a FOR command cannot be piped to a filter, so it is impossible to use the MORE command, for example, to pause the display of files typed to the screen when the screen is full. However, the output of the FOR command can be redirected to a file. The above command could be rewritten to send the output of the TYPE commands to a file named TEMP.FIL:

```
FOR %A IN (*.BAT) TYPE %A > TEMP.FIL
```

If Doskey is installed, then multiple commands may be entered on one line. Thus, the above command could be expanded further, as follows:

```
FOR %A IN (*.BAT) DO TYPE %A >> TEMP.FIL ¶
TYPE TEMP.FIL | MORE
```

The FOR command first writes the contents of all the batch files to the file named TEMP.FIL (recall that the >> causes output to be appended to a file, rather than overwriting it). The ¶ (paragraph mark) indicates that a new command follows; the second TYPE command then displays the contents of TEMP.FIL, but pipes the output through the MORE filter to cause the display to pause whenever the screen is full. Remember that Doskey must be loaded in order to use the paragraph mark to put multiple commands on the same line. A paragraph mark is entered on the command line by pressing Ctrl-T.

FORMAT

Floppy disks usually need to be formatted before they can be used. Formatting organizes a disk so that DOS will know how to use it. Hard disks also need formatting, but this is usually done by the computer manufacturer or dealer before the computer is sold, so the FORMAT command is usually used only with floppy disks. To format a disk in drive A: enter the following command:

```
FORMAT A:
```

DOS will prompt you to insert a disk in the A: drive and press Enter↵ when ready. It will then format the disk, display information about the size of the disk, and display messages regarding the results of the FORMAT operation. There are several switches that may be used with FORMAT, which provide control over certain aspects of how a disk is formatted. When FORMAT is done, it asks if you want to Format another disk. You may either choose Y, for Yes, in which it case you are prompted to put another disk in the drive; or N, for No, in which case you are returned to the command prompt. FORMAT is discussed in detail in Chapter 6.

RENAME

The RENAME command is used to rename a file. The syntax is:

```
RENAME name1 name2
```

where **name1** is the current name of the file, including the extension, and **name2** is the new name of the file, also including the extension. A path may be specified for name1, but cannot be specified for name2. This is because the path specified for name1 identifies the location of the file, and the location has to be the same for name2. If a file with the name2 name already exists in the specified directory, the message "Duplicate file name or file not found" will be displayed and the file will be left unchanged.

TYPE

TYPE is used to type the contents of a text file to the screen. If the file contains anything other than plain text characters, the non-text characters will appear as odd symbols on the screen. These non-text characters are typically used by word processors to indicate special formatting commands — to print text in boldface or underlined, for example. To display a text file named SAMPLE.TXT on the screen, the following command is used:

```
TYPE SAMPLE.TXT
```

The TYPE command is commonly used in conjunction with the MORE filter, in order to prevent text from scrolling off the top of the screen before you have a chance to read it.

VER

The VER command displays the version number of DOS that is being used. If you are using DOS 5.0 and enter the VER command, the following message will display on your screen:

```
MS-DOS Version 5.0
```

Customizing the Command Prompt

The default command prompt is simply the current drive letter, followed by a greater-than sign. For example, if the current drive is C, then the default command prompt is:

```
C>_
```

This prompt provides a minimal amount of information: it tells you what the current drive is, but no more. The PROMPT command is used to modify the command prompt, to provide more useful information than just the drive letter. It is common practice to place a **PROMPT PG** command in the AUTOEXEC.BAT file, to define a prompt that indicates the current directory as well as the current drive. If the current drive is C, and the current directory is \WP\LETTERS, then this prompt will appear as:

```
C:\WP\LETTERS>_
```

For an experienced user, this is a meaningful prompt. However, for people who are not familiar with DOS, this prompt is fairly cryptic. The PROMPT command allows you to design a prompt that will be suitable for almost any circumstances. Among other things, you may use a text message for the prompt, and you may embed the current date, time, drive, directory, and other information, in the prompt. For example, to create a prompt that says "Enter your command here: ", the following command is used:

```
PROMPT Enter your command here:
```

There are thirteen parameters that may be used with the PROMPT command (a complete list can be found in Chapter 10). Each parameter is preceded by a dollar sign ($). The dollar sign is used to tell DOS that the character which follows it should be interpreted as a parameter, rather than a text character. The parameters allow you to embed a variety of items in the prompt. Examples in this book always use upper case letters for the Prompt parameters, but lowercase letters work just as well. You may intersperse text before, between, and after parameters. If you want to reset the DOS Prompt to its default value (the current drive letter followed by a greater-than sign), you may do so by entering the PROMPT command without parameters.

Including a Dollar Sign in the Prompt

Since a single dollar sign is used to indicate that the following character should be interpreted as a parameter, it cannot be used to simply display a dollar sign in the prompt. To include the dollar sign character in a prompt, two dollar signs are used. For example, to change the prompt to "Enter your command $ ", the following command is used:

```
PROMPT Enter your command $$
```

Changing the DOS Prompt to a Text Message

You may change the prompt to a message, without any drive or directory designators, by entering the PROMPT command followed by the text you want to use. If you enter the following command:

```
PROMPT Type your DOS command here:
```

The DOS prompt will look like this:

```
Type your DOS command here:_
```

If you want a space to appear at the end of the prompt, before the cursor position, then you should leave a space at the end of the prompt message when you enter the PROMPT command.

Including a Greater-than Sign (>) in the DOS Prompt

It has been a standard convention since the first version of DOS to end the prompt with a greater-than sign. However, since the greater-than sign is the DOS output redirection symbol, you may not use it directly in a text message. If you include it in the text of your prompt, you will get the error message "File creation error" because DOS will think you are trying to redirect the output of the PROMPT command to a file. The $G parameter is therefore used to include a greater-than sign in the prompt. To replace the colon with a greater-than sign in the prompt described above, the following command would be used:

```
PROMPT Type your DOS command here$G
```

This will result in the following prompt:

```
Type your DOS command here>_
```

Including a Less-than Sign (<) in the DOS Prompt

The less-than sign is the DOS input redirection symbol, so it too requires a parameter to be included in a prompt message. The $L parameter is used to place a less-than sign in a prompt. To embed text between less-than and greater-than signs you enter the text between the $L and $G parameters. To make the prompt appear like this:

```
Type your < DOS > command here:_
```

the following command would be used:

```
PROMPT Type your $L DOS $G command here:
```

Including the Current Drive Letter in the DOS Prompt

The current drive letter can be included in the prompt with the $N parameter, as in the following PROMPT command:

```
PROMPT The current drive is: $N
```

If your current drive is C this will result in the following prompt:

```
The current drive is: C_
```

It is helpful to terminate any prompt message with a symbol, so that it is clear where the prompt ends and a command that has been entered begins. This is why the greater-than sign developed as a standard terminator for DOS command prompts. The above prompt command is improved by adding the $G parameter:

```
PROMPT The current drive is: $N$G
```

This produces the following prompt:

```
The current drive is: C>_
```

Including the Current Directory Path in the DOS Prompt

The $P parameter is used to insert the path from the root directory to the current directory into the DOS prompt, as in the following command:

```
PROMPT The current directory is $P:
```

If you are in the C:\DOS directory this will produce the following prompt:

```
The current directory is C:\DOS:_
```

Notice that the path includes the drive letter so there is no need to combine the $N and $P parameters. In fact, if you used NP in a prompt you would have the drive letter duplicated at the beginning of the prompt.

One of the most common PROMPT commands is simply:

```
PROMPT $P$G
```

This prompt is usually installed in the AUTOEXEC.BAT file, making it the prompt that automatically appears when the computer is booted. If you are in the DOS subdirectory this prompt will look like this:

```
C:\DOS>_
```

Note: When the $P parameter is used, it causes DOS to read the directory of the disk in the current drive after each command. This is necessary because some commands change the current directory. Since $P displays the directory name, it must check after each command to see which directory is current. With floppy drives, reading the directory after each command significantly slows down DOS's response time. Consequently, if you frequently use a floppy drive as the current drive, you may want to set a prompt that does not use the $P parameter.

An additional problem can arise when using $P with a floppy drive. In order for DOS to read the directory of a floppy drive, there must be a formatted disk in the drive. Consequently, if you are going to use a floppy drive as the current drive with the $P parameter, you must be sure the drive has a formatted disk in it. If it doesn't, DOS will display the following message:

```
Not ready reading drive A
Abort, Retry, Fail?
```

There are two ways of responding to this message: the first is to put in a formatted floppy disk and select either A (Abort) or R (Retry). Either one will result in the drive being read and the correct prompt appearing. The other option is to select F (Fail). This will result in the following message:

```
Current drive is no longer valid>
```

You do not need to worry about this message — it does not disable the drive, but allows you to take whatever appropriate action you like. Possible actions are:

1. Insert a formatted disk in the floppy drive and press Enter↵. This will cause DOS to read the drive again and set the prompt properly.
2. Change the current drive to one that has a valid disk in it — either another floppy drive or a fixed disk drive. To change to your C: (fixed) drive, enter C:. This will result in drive C becoming the current drive and the DOS prompt will display the current directory on drive C.

Including the Date in the DOS Prompt

The $D parameter is used to insert the system date into the DOS prompt. The following command demonstrates the use of the $D parameter:

```
PROMPT Today's date is $D. Enter your command$G
```

This produces the following prompt:

```
Today's date is Wed 01-24-1989. Enter your command>_
```

Since DOS stores the date of last modification with each file name, it is important to have the system date set properly. Most computers today have battery powered clocks that maintain the system date and time even when the computer is turned off, but under some circumstances these clocks can lose their setting. Including the date in the prompt is one way of being sure you know what the system date is. It also provides a useful way of checking today's date — simply look at your computer screen.

Including the Time in the DOS Prompt

The system time may be inserted into the DOS prompt using the $T parameter. When the time appears in the command prompt, it will always be displayed in 24 hour format. The following command demonstrates the use of the $T parameter:

```
PROMPT The time is $T. Enter your command$G
```

This results in the following prompt (assuming it's 2:06:17.35 PM):

```
The time is 14:06:17.35. Enter your command>_
```

You may not want the entire time displayed, out to hundredths of a second. The $H parameter, discussed below in the section "Using a Backspace in the DOS Prompt," can be used to truncate numbers off the time.

Including the DOS Version Number in the DOS Prompt

If you are running different versions of DOS on your computer, or have more than one computer with different versions of DOS, it can be useful to know what version of DOS is running. This can be shown with the $V parameter. Both the version number and the manufacturer (IBM or Microsoft) is shown. The following command demonstrates this:

```
PROMPT $V
```

If you are running MS-DOS 5.00, this produces the following prompt:

```
MS-DOS Version 5.00_
```

Including a Vertical Bar (Pipe) Symbol in the DOS Prompt

Like the greater-than and less-than signs, the vertical bar is a DOS redirection symbol, and therefore cannot be used directly in a prompt message. To include a vertical bar in the DOS prompt, the $B parameter is used. The following command demonstrates the use of the $B parameter:

```
PROMPT Enter your command here$B
```

This will produce the following prompt:

```
Enter your command here|_
```

Including an Equal Sign in the DOS Prompt

In some versions of DOS, there have been circumstances in which the equal sign (=) could not be used directly in a prompt message. It was therefore necessary to define a special parameter to specify an equal sign. The equal sign parameter is $Q. The following command demonstrates this:

```
PROMPT Today's date $Q $D$G
```

This produces the following prompt:

```
Today's date = Wed 01-24-89>_
```

In DOS 5.0, the following PROMPT command works just as well:

```
PROMPT Today's date = $D$G
```

Using a Backspace in the DOS Prompt

You may include a backspace character in your PROMPT command with the $H parameter. The effect of this is to erase the character immediately preceding the $H. For example, if you used the following command:

```
PROMPT This demonstrates the use of a b$Hackspace.
```

the prompt would look like this:

```
This demonstrates the use of a backspace._
```

At first sight, this seems to be a useless parameter — why display a character you're just going to erase? But remember that some of the other parameters display whole strings of characters as part of the prompt. For example, the $T (time) parameter displays the time as hh:mm:ss.hs. You may prefer to have the time displayed only as hours and minutes. The last six characters of the time can be erased using the $H parameter, as in the following command:

```
PROMPT The time is $T$H$H$H$H$H$H$G
```

If the system time is 10:04:02.01, the prompt will appear as:

```
The time is 10:04>_
```

Displaying More than One Line in the DOS Prompt

As the DOS prompt gets longer, it gets harder to distinguish the different pieces of information, and there is less room for you to type in your commands and keep them on one line. DOS will allow you to wrap your commands to subsequent lines, but they are harder to read. The $_ parameter allows you to break your prompt into separate lines. Take, for example, the following command:

```
PROMPT The current directory is $P. The date is $D. The time is $T.
```

This results in a prompt that is so long it actually wraps to the next line:

```
The current directory is C:\. The date is Sat 06-01-1991. The
time is 14:24:38.48.
```

You can make this prompt more readable by using the $_ parameter to separate the three statements, as follows:

```
PROMPT The current directory is $P.$_The date is $D.$_The time is $T.
```

This will produce the following prompt:

```
The current directory is C:\DOS.
The date is Sat 06-01-1991.
The time is 14:24:38.48._
```

Changing the Color of the Prompt

The PROMPT command can be used to provide extensive control over the colors and positioning of text on the screen. This use of the PROMPT command is discussed in detail in Chapter 9. A limited application of these techniques, to change the color of the prompt itself, is presented here.

In order for this technique to work, the following command must appear in your CONFIG.SYS file:

```
DEVICE=ANSI.SYS
```

If the ANSI.SYS file is not in the root directory of the boot disk, then the path to the file must also be specified. The most common version of the command is:

```
DEVICE=C:\DOS\ANSI.SYS
```

If this command does not appear in the CONFIG.SYS file, you may add it, using a text editor. For this command to take effect, you must reboot your computer after adding it to the CONFIG.SYS file. Once you have rebooted with the ANSI.SYS driver loaded, the following command will create the standard command prompt, but it will appear in reverse video:

```
PROMPT $E[7M$P$G$E[M
```

There are three parts to this prompt:

- a command to the ANSI.SYS driver **$E[7M**
- the normal PROMPT parameters **PG**
- the second command to ANSI.SYS **$E[M**.

The first ANSI.SYS command tells DOS to start using a reverse video display, the PG displays the current directory and the greater-than sign, and the final ANSI.SYS command tells DOS to stop using the reverse video display.

Any of the standard prompt parameters can be used in addition to, or in place of, the PG shown above. There are many different commands that can be sent to ANSI.SYS, allowing you to position text in various areas of the screen, display text and prompt messages in different colors, and affect the way the keyboard works. All of these are discussed in detail in Chapter 9.

Using the Doskey Command Stack

SHORTCUT

Doskey is a program that provides three features for streamlining your work at the command prompt. The first feature is the ability to enter multiple commands on the same line. The second feature is a "command stack," which remembers previous commands that have been entered. These commands can then be retrieved, edited, and reused, without being retyped. The command stack is cleared whenever the computer is shut off. The third feature is a "macro" facility. A macro is a brief shortcut for a long sequence of keystrokes. With Doskey, you can create your own macros, allowing you to quickly execute frequently used commands and sequences of commands. Macros can be saved in disk files and reloaded after the computer has been shut off. Doskey also extends the use of the ←BkSp key, making it possible to ←BkSp over multiple lines on the screen that together make up one command line.

Doskey is known as a **T**erminate and **S**tay **R**esident (TSR) program. When you run Doskey, it loads itself into memory, then allows you to go about using the computer for your usual tasks. However, it is always there in the background, memorizing everything you enter at the command line, and providing macro capabilities. Because it is a TSR, and therefore resident in the computer's memory, Doskey takes a small amount of memory away from your other programs. The amount of memory used by Doskey, as well as other variations on how Doskey works, can be controlled with switches when Doskey is started. These switches are discussed later in this chapter, in the section titled, "The Doskey Command Syntax." A technical discussion of how Doskey works is presented in the Advanced Topics section of this chapter. For now, simply start Doskey without any switches by entering the command DOSKEY at the command prompt.

Entering Multiple Commands on One Line

Without Doskey, you are limited to entering one command at a time. With Doskey loaded, however, you may enter a string of commands that will be executed in sequence. The only limitation is that the entire string cannot exceed 127 characters.

Multiple commands on the same line must be separated with a paragraph (¶) mark. The paragraph mark is inserted into the command line by holding down Ctrl and pressing T. No spaces are required between the paragraph mark and the commands, but spaces make the command line more readable. As DOS processes the commands, it echoes each command name back to the screen before it begins to execute.

Recalling Previous Commands

As soon as Doskey is loaded, it starts remembering all commands entered at the COMMAND PROMPT by placing them in the "command stack." The command stack can be thought of as a stack of DOS command lines. The first command line entered after Doskey is loaded is placed on the bottom of the stack. The next command line is then pushed onto the stack beneath the first one, in effect pushing the first one up. This continues until the command stack is filled up. When the stack is full, and another command line is entered, the command line at the top of the stack, which is the oldest command, is removed and the new one is pushed onto the bottom. The stack is thought of as containing command lines, rather than just commands, because with Doskey loaded, multiple commands can be entered on one line. When this is done, Doskey stores the entire line as one item in the stack, not each individual command that appears on the line. For the rest of this discussion, the terms "command" and "command line" will be used interchangeably.

Unless you started Doskey with the /BUFSIZE switch Doskey will reserve 512 bytes of memory for holding the command stack and macros. Each character in a command occupies one byte, so 512 bytes permits a total of 512 characters to be stored. If the average command is eight characters, then Doskey can store 64 commands. If no macros are being used, then the command stack will have access to all 512 bytes, allowing the last 64 commands to be remembered (assuming an eight character average). If 128 bytes are being used by macros, then 384 bytes will be available to the command stack, which means the last 48 commands can be recalled. Macros are discussed in the next section.

There are several keys used for recalling previous commands from the stack, making it easy to quickly find the command you want. For the examples that follow, the command stack shown in Figure 3-4 will be used.

```
1: CD \WP\LETTERS
2: FOR %F IN (*.LET) DO FIND "Southern Region Sales" %F
3: DIR A*.LET
4: COPY A*.LET C:\BACKUP\LETTERS
5: DEL A*.LET
6: CD C:\BACKUP\LETTERS
7: DIR A*.LET
8: CD \
9: DIR
```

Figure 3-4 — Sample Doskey Command Stack.

The blank command line on which you enter new commands can be thought of as a blank space beneath the stack (line 10 in this example). As new commands are entered, they are pushed into the bottom of the stack. The ↑ is used to recall the previous command from the stack. For example, after the last command in Figure 3-4 has been executed (DIR), a blank command line will be displayed. Pressing ↑ once displays the DIR command (number 9 in the stack). The DIR command is not executed, it is only displayed. To execute it, press Enter↵. If you do not press Enter↵, then pressing ↑ again will display the CD \ command (number 8 in the stack), and so on. The ↓ is used to move down the stack (towards the newest command). The ↑ is used to move up the stack (towards the oldest command). Eventually, pressing ↓ will display the blank command line below the bottom of the stack.

↑ and ↓ are useful for moving through a few lines in the stack, but when there are a lot of commands to move through they are not very efficient. The PgUp and PgDn are quicker. PgUp takes you to the top of the stack, where the oldest command is. PgDn takes you to the bottom, or most recent, command.

F7 can be used to display a numbered list of all the commands in the stack. F7 was used to produce the list in Figure 3-4. This is useful for reminding you of what is in the stack. F7 can also be used in conjunction with the F8 and F9 keys.

F8 is used to cycle through a subset of the commands in the stack. The subset is defined as those commands that begin with specified characters. For example, if the blank command line beneath the bottom of the stack in Figure 3-4 is displayed, pressing C, then pressing F8 would display line 8, `CD \`. Pressing F8 again would display line 6, `CD C:\BACKUP\LETTERS`. Pressing F8 again would display line 4, and so on. F8 treats the stack as a circular queue, so that when the specified command at the top of the stack is reached, pressing F8 again goes to the first specified command at the bottom of the stack. In this example, starting at the bottom of the stack with the letter `C` and pressing F8 four times would display line one in the stack, `CD \WP\LETTERS`. Pressing F8 again would display line eight, `CD \`.

When F8 is pressed following a search string, the cursor stays in the same position when selected commands are displayed. Using the command stack in Figure 3-4, if the letter `C` were the search string and the currently displayed command were `COPY A*.LET C:\BACKUP\LETTERS`, the cursor would be beneath the `O` in COPY. F8 uses everything prior to the cursor as the search string. It can therefore be used for increasingly selective searches. For example, pressing C and F8 once displays the command `CD \`, with the cursor blinking beneath the `D`. Pressing end moves the cursor to the end of the command line. Pressing F8 again will display line 1, `CD \WP\LETTERS`, which is the next previous command that begins with `CD \`.

F9 is used to jump to a command based on its line number in the command stack. (Use F7 to review the line numbers of the command in the stack.) When F9 is pressed, Doskey displays the "Line number: " prompt. Entering a line number and pressing Enter← jumps to that line in the stack, the command in that line is then displayed on the command line. In the command stack shown in Figure 3-4, pressing F9 and entering the number 4 would display the

```
COPY A*.LET C:\BACKUP\LETTERS
```

command. Entering a 0 or 1 takes you to the oldest command in the stack; entering any number greater than or equal to the newest command takes you to the bottom of the stack. Doskey permits up to five digits, so it can reference up to 99999 lines. Doskey permits only numeric keys to be typed at the "Line number: " prompt; any other keys are ignored.

When a particular command from the stack is displayed, you are positioned at that line in the stack. In other words, you may use F8 or F9 to jump to a location in the stack, then use the arrow keys to move away from that point one command at a time.

When a line from the stack is displayed and Enter↵ is pressed, Doskey does not execute the original copy of the command. Instead, it moves to the bottom of the stack and inserts a new copy there, which it then executes. Consequently, the commands in the stack are always left unchanged, and whenever a command is executed you are again placed at the bottom of the stack. If you want to replay two or three lines from the same part of the stack, you will have to return to that part of the stack for each command. F9 can be particularly useful in this situation.

esc clears a command from the screen, without changing your location in the stack or affecting the command in the stack. For example, with the stack shown in Figure 3-4, if the currently displayed command were line 3, `DIR A*.LET`, pressing esc would display a blank command line. Pressing ↑ would display line 2, and pressing ↓ would display line 3 again. If you typed a new command and pressed Enter↵, that command would be placed at the bottom of the stack.

Keys Used for Editing the Displayed Command

As discussed earlier in this chapter, when Doskey is not loaded, the last command entered is stored in an area of memory known as the "template." The function keys used for editing the template — F1 through F5 — are of limited use when Doskey is loaded, because they are only effective immediately after a command has been entered. When you display a command from the stack, the contents of the template are replaced with the contents of the line in the stack. This can be demonstrated by displaying a command from the stack, then using ←BkSp to erase it, then pressing F3 to play back the command. This shows that the template contains that command. Consequently, if you use ↓ to return to the blank line beneath the stack, the template will be filled with blanks. Doskey does not change the function of the F6, Delete, Insert, and Escape keys when it is loaded, because these keys are implemented as part of Doskey to work on any displayed command.

Doskey implements several keys to provide editing capabilities on displayed commands. The function of these keys is listed in Table 3-4.

Table 3-4 — Keys Used for Recalling and Editing the Command Stack.

Key	Effect
←	Moves the cursor one character to the left.
→	Moves the cursor one character to the right.
Ctrl-←	Moves the cursor one word to the left.
Ctrl-→	Moves the cursor one word to the right.
home	Moves the cursor to the beginning of the line.
end	Moves the cursor to the end of the line.
←BkSp	Deletes the character to the left of the cursor.
delete	Deletes the character at the cursor.
Ctrl-home	Deletes all characters from the beginning of the line up to but not including the cursor.
Ctrl-end	Deletes all characters from the cursor to the end of the line. The character at the cursor is deleted.
ins	Toggles between Insert mode and Overstrike mode.
esc	Clears the displayed from the screen.

There are six keys used for moving the cursor around on the command line: ←, →, Ctrl-←, Ctrl-→, home, and end. All of these affect the cursor position only — moving them does not change the command line itself. For the Ctrl-← and Ctrl-→ keys, a word is defined as whatever appears between two blank spaces.

Four keys are used for deleting characters: ←BkSp, delete, Ctrl-end and Ctrl-home. When Ctrl-end is used, the character at the cursor is included in those deleted. When Ctrl-home is used, the character at the cursor is not deleted.

Ins is used to toggle between Insert mode and Overstrike. Insert mode means that anything that is typed is inserted into the line, moving everything following the cursor position to the right. Overstrike mode means that anything that is typed replaces what is at the cursor. By default, Doskey is in Overstrike mode, unless Doskey is started with the /INSERT switch.

esc clears the displayed command from the screen. It does not, however, remove the command from the command stack. If a line from the stack is displayed, then esc is used to clear it from the screen, pressing ↑, then ↓ will restore it to the screen because it still exists in the stack.

Sometimes the command stack will become cluttered with lots of commands you don't want. The entire stack can be cleared by pressing alt-F7.

Creating Batch and Text Files from the Command Stack

The commands in the stack can be stored in a text file. There are two common reasons for doing this: to remind yourself later what sequence of commands you have entered; and to create a batch file, which can be invoked to replay the commands at any time. Batch files are text files which contain DOS commands and have a name extension of BAT. Batch files are discussed in Chapter 8.

The /HISTORY switch is used with the Doskey command to display the command stack without the line numbers that are produced by F7. For example, if the command stack contained the commands shown in Figure 3-4, then entering the command `DOSKEY /HISTORY` would display the following:

```
CD \WP\LETTERS
FOR %F IN (*.LET) DO FIND "Southern Region Sales" %F
DIR A*.LET
COPY A*.LET C:\BACKUP\LETTERS
DEL A*.LET
CD :\BACKUP\LETTERS
DIR A*.LET
CD \
DIR
DOSKEY /HISTORY
```

Since this list appears without line numbers, it constitutes a set of executable commands. Now the only problem is getting them into a file. This can be accomplished by redirecting them to a file using the greater-than character (>). Redirection is discussed in detail earlier in this chapter.

```
DOSKEY /HISTORY > MYFILE.BAT
```

will create a file named MYFILE.BAT with the above list of commands in it. Now, entering the command MYFILE will execute those commands, one at a time.

Notice that the last command entered, DOSKEY /HISTORY, is included in the displayed command stack. Usually this is not wanted in a batch file, so it is necessary to edit the file with a text editor to delete this line. The text editor must be capable of saving files in plain ASCII format. The MS-DOS EDITOR that comes with DOS 5.0 is a good choice.

Doskey can be an efficient way to create batch files, since it allows you to test the commands as you go before putting them into the batch file. The following steps will accomplish this:

1. Load Doskey (if it is not already running).
2. Press alt-F7, to clear the command stack. This step is not necessary if you have just loaded Doskey, since the command stack is already empty.
3. Enter each of the commands you want to include in the batch file.
4. Use the `DOSKEY /HISTORY > FILENAME.BAT` command to save the commands in a batch file named FILENAME.BAT.
5. Edit FILENAME.BAT to remove the following line from the end of the file:

```
DOSKEY / HISTORY > FILENAME.BAT
```

Creating and Using Macros with Doskey

The macro feature of Doskey allows you to create “mini-commands” that can replace long commands or sequences of commands. For example, a macro named LD (for Letters Directory) could be defined to store the command

```
DIR C:\WP\LETTERS\*.LET
```

Typing LD and pressing Enter← would then execute this command.

Doskey’s macro feature is quite powerful, allowing you to write macros that accept parameters, take input from files or devices instead of the keyboard, and send output to files or devices instead of the screen. Macros are stored in memory, so they execute very quickly.

Creating Macros

Macros are created using the following version of the Doskey command:

```
DOSKEY macro=commands
```

Where **macro** represents the characters that will be typed to execute the macro, and **commands** represents one or more commands that will execute when the macro is run. A simple example is a macro to replace the **DIR /P** command with the single letter D. (The /P switch causes the directory listing to pause after each screen of information and wait for a key to be pressed). The Doskey command to create this macro would be:

```
DOSKEY D=DIR/P
```

If Doskey is not already installed, this command will install it and then create the macro. If Doskey is installed, the macro will be created without installing a second copy of Doskey.

Doskey macros can only be used for commands that are entered on the command line. Once you enter a software program, Doskey macros are ignored. There are programs available that will let you define macros within any program. These programs generally provide much more power than Doskey in terms of the size and complexity of the macros, as well as the ability to easily edit macro definitions. A number of software packages also have their own built-in macro features. A few examples are Lotus 123, Microsoft Excel, WordPerfect, and Microsoft Word.

Running Macros

Once a macro is created, it behaves just like a DOS command — typing the name of the macro and pressing Enter← causes the macro to execute. DOS is not case sensitive; you may type the macro name in any combination of upper and lower case letters you like, and DOS will find it.

What is happening behind the scenes when a macro is executed is that a macro interpreter looks up the command stored in the macro and sends that command to the DOS command line. To DOS, then, the effect is that the command was typed in and Enter← was pressed. You can see the command echoed back on the screen before it executes. However, the command stack will not contain the command itself, it will contain the macro.

There is one difference between how macros and DOS commands may be entered on the command line: DOS commands can be entered with leading blanks, while macros may not — they must start in the first position of the command line. For example, the DIR command could be entered as

```
DIR
```

or as

```
    DIR
```

LoOk

Either of these is acceptable; both will produce the same listing of the files in the current directory. However, if a macro named XXX were entered as

```
    XXX
```

it would not execute. DOS would display the message "Bad command or file name," because it would not be able to find a program file or command name that matched.

When DOS 5.0 is given a command, it first checks all macros to see if one matches the command. If a matching macro is found, DOS executes the macro. If no matching macro is found, DOS then checks its internal commands, and finally it looks on the disk for a program that matches. The difference between how leading blanks are handled for macros and for DOS commands can be exploited to create a macro with the same name as a DOS command, and still have both available. This is useful when a DOS command is nearly always run with the same set of parameters and/or switch settings: a macro can be defined that contains the parameters and switch settings. The macro will execute when the command is entered with no leading blanks, and the DOS command will execute when entered with one or more leading blanks. For example, you may want the DIR command set up to use the WIDE format. This can be accomplished with the following macro:

```
DOSKEY DIR=DIR /W
```

When this macro has been created, entering DIR with no leading blanks will display the current directory in wide format, while entering DIR with one or more leading blanks will execute the usual form of the DIR command. By creating a macro with the same name as a DOS command, you only have to remember a single command, but can execute it in two different ways.

Note that in the macro version of the DIR command described here, there is no way to specify a directory — the macro always displays the current directory. By adding a replaceable parameter, discussed later in this chapter, you can create a DIR macro that still uses wide format, but also accepts a directory specifier.

Creating and Using Macros with Multiple Commands

The macros discussed so far contain only a single command. Macros that contain multiple commands can also be created. To define a macro with more than one command, the commands must be separated by the $T or $t symbol. Without a special separator, DOS would not know whether to interpret a word as a parameter to the previous command, or as a separate command. The following macro definition demonstrates:

```
DOSKEY LD=CD \WP\LETTERS $T DIR /W
```

When this macro is executed, DOS will change directories to the \WP\LETTERS directory, then display that directory in wide format. The space between the $T symbol and the commands is optional.

The only limitation that is imposed on how many commands can be placed in one macro is that the total length of the macro must not exceed 127 characters. This includes blank spaces and special symbols, such as the $T symbol shown here, which counts for two characters. To conserve space, the above command could be written as:

```
DOSKEY LD=C:\WP\LETTERS$TDIR/W
```

Using Replaceable Parameters in Macros

Many commands accept parameters — values that can be passed to them from the command line when they execute. To make macros more useful, the ability to include parameters in macros is provided. When a parameter is included in a macro, the parameter is specified with a special symbol, which indicates where it should be inserted into the macro. The symbols are $1 through $9. For example, the following command creates a macro (WD) that contains one parameter:

```
DOSKEY WD=DIR $1 /W
```

When this macro is run, a path should be specified on the command line to tell the macro what directory to display. The path that is specified will be passed to the DIR command as a parameter. For example, to do a DIR/W on the WP\LETTERS subdirectory, enter:

```
WD C:\WP\LETTERS
```

When this command is entered, the macro interpreter looks up the command that corresponds to the macro, plays it back on the command line, and inserts the parameter wherever it encounters a $1 symbol. Thus, what would be echoed to the screen for this example is:

```
DIR C:\WP\LETTERS /W
```

The same parameter may be used multiple times in the same macro. For example, a macro named COPYDEL could be written to copy one file to another file, then to delete the first file:

```
DOSKEY COPYDEL=COPY $1 $2 $T DEL $1
```

The parameter symbols do not have to be used in sequence — $2 may appear before $1 in the macro — but if a parameter symbol is used, then all parameter symbols less than that one must also be used. For example, if $4 is referenced, then $1, $2, and $3 must also be referenced somewhere in the macro. Remember, too, that $2 always refers to the second parameter that is specified when the macro is run, regardless of where $2 appears in the macro itself.

Macros that use the replaceable parameters $1 through $9 will only accept as many parameters as the macro specifies. For example, the macro created by the following command requires that only one parameter be used:

```
DOSKEY C=COPY $1
```

This macro abbreviates the COPY command down to one letter, and copies the specified file into the current directory. It will only allow the source file to be specified — the target is always assumed to have the same file name as the source, and be in the current directory. The COPY command itself, however, can be run with two file names, to specify both a source and target. This allows you to rename the target file during the copy operation. There are also optional switches that can be specified for the COPY command. For example, if C: is the current drive, then the following command will copy the file LETTER1.FIL in the current directory to the file LETTER2.FIL, also in the current directory, and verify the copy:

```
COPY LETTER1.FIL LETTER2.FIL /V
```

If a second parameter, specifying a target file name, and a switch, were used the C macro defined above, the second parameter and switch would be ignored. The target file would have the same name as the source file, and the switch would not have any effect.

The $* parameter symbol provides more flexibility in the use of parameters with macros. This symbol means "take everything that follows the name of the macro and append it to the command." Since DOS interprets spaces as being separators between parameters, you can now include as few or as many parameters as you like. The C macro discussed above could be defined as follows:

```
DOSKEY C=COPY $*
```

This macro will now accept whatever parameters and switches you want to pass to it. Both of the following commands would execute properly:

```
C A:\LETTER1.FIL C

LETTER1.FIL LETTER2.FIL/V
```

Redirecting Macro Input and Output

Earlier in this chapter, redirecting the input and output of commands entered at the command prompt was discussed. Redirection symbols may be embedded as part of the command(s) stored in a macro. However, the symbols used for redirection within a macro are different from those used at the command prompt. Macro redirection symbols and their command line equivalents are shown in Table 3-5.

Table 3-5 — Macro Redirection Symbols.

Macro Symbol	Equivalent Command Line Symbol	Effect
$B or $b	|	Pipes the output from one command to the input of another.
$G or $g	>	Redirects the output.
GG or GG	>>	Appends the output to the specified file.
$L or $l	<	Redirects the input.

Note: It is important to understand that these symbols are only used on the commands WITHIN the macro. For example, the following command creates a macro named DM, that pipes the output of the DIR command to the input of the MORE filter:

```
DOSKEY DM=DIR $B MORE
```

When the DM macro is executed, the directory listing of the current directory is displayed, with a pause after each screen. Redirection symbols cannot be used on the macro itself. Consider the following macro definition:

```
DOSKEY D=DIR
```

Entering the command D will now display the current directory. However, in the following command, the Pipe symbol and MORE filter are ignored:

```
D | MORE
```

Parameters and redirection symbols can be combined in one macro. For example, the following DOSKEY command creates a macro named FL (for FILE LIST) that scans the current directory for all file names that contain a specified string, and displays the list of those file names one screen at a time:

```
DOSKEY FL = DIR $B FIND "$1" $B MORE
```

When this macro is run, a single parameter must be specified with it. The parameter contains the text string to search for in file names. Since double quotes are included in the macro, they are not needed on the parameter. For example, to display all file names in the current directory that contain the characters LET, the following command would be used:

```
FL LET
```

The first $B in the macro pipes the output of the DIR command into the input of the FIND command. DOS substitutes the parameter from the command line for the $1 symbol. The second $B pipes the output of the FIND command into the input of the MORE command, which pauses the screen after each screen of file names is displayed. This macro will find all files that have the letters LET anywhere in their name, including the extension.

Using the Parameter and Redirection symbols, and the filter commands, it is possible to develop sophisticated macros for automating tasks that would otherwise be quite tedious. Two examples appear at the end of the following section.

Listing, Deleting, and Saving Macro Definitions

Macros are very useful tools for increasing the efficiency with which you can use the computer. In order to manage the macros you use, DOS 5.0 provides ways of listing them, deleting them, and saving their definitions in a file. To list all macros that are currently defined, enter:

```
DOSKEY /MACROS.
```

The macro definitions will be displayed on the screen.

To delete a macro, enter the macro definition command, but leave the line blank after the equal sign. For example, to delete a macro named WDIR, use the following command:

```
    DOSKEY WDIR=
```

Macro definitions can be saved by using the /MACROS switch and the output redirection symbol (>). Suppose these four macros were defined:

```
DIR DIR $1 /W

WRITECD     \WP\LETTERS $T WP $1

LOTUS  CD\123\DATA $T 123

BU  BACKUP C:\WP\LETTERS\*.* A:
```

The following command saves all of the existing macro definition in a file named MACRO.BAT:

```
    DOSKEY /MACROS > MACRO.BAT
```

The MACRO.BAT file will contain the following:

```
DIR=DIR $1 /W

WRITECD=\WP\LETTERS $T WP $1

LOTUS=CD\123\DATA $T 123

BU=BACKUP C:\WP\LETTERS\*.* A:
```

By editing this file to put the word DOSKEY at the start of each line, you can turn it into a macro initialization file. When this file is executed, all of the macros will be created. To edit the file, be sure to use a text editor that can save files in plain ASCII format. The MS-DOS Edition that comes with DOS 5.0 is a good choice.

A macro that will automatically store all your macros in a specified file, in sorted order, is:

```
MAKEMAC=DOSKEY/MACROS $B SORT $G $1
```

The $B pipes the output of the DOSKEY/MACROS command to the input of the SORT filter; the $G redirects the output of the SORT filter to a file specified on the command line, which is substituted for the $1 parameter. If the four macros shown above were defined, as well as the MAKEMAC macro, then the following command would create a file named MACS.BAT:

```
MAKEMAC MACS.BAT
```

The contents of MACS.BAT would be:

```
BU=BACKUP C:\WP\LETTERS\*.* A:

DIR=DIR $1 /W

LOTUS=CD\123\DATA $T 123

MAKEMAC=DOSKEY/MACROS $B SORT $G $1

WRITECD=\WP\LETTERS $T WP $1
```

To keep a master list of all macros you ever defined, the following macro appends the current macro definitions to an existing file, then sorts that file. Note that in this macro, the same file is used for both the input and the output of the SORT filter:

```
ADDMAC = DOSKEY/MACROS $G$G $1 $T SORT $L $1 $G $1
```

The GG $1 symbols redirect the output of the DOSKEY/MACROS command to be appended to the file specified on the command line. The $T symbol separates the SORT command from the DOSKEY/MACROS command. The $L $1 symbols redirect the input of the SORT command to be the file specified on the command line, and the $G $1 symbols redirect the output of the SORT command to overwrite the file specified on the command line. If ADDMAC is used to add macro definitions to a file that already contains some of the existing macros, then those macros will be duplicated in the file. However, duplicate macros are easy to spot, since sorting the file puts them next to each other.

Using Dollar Sign Symbols in Macros

As seen above, the dollar sign symbol ($) is used to define parameters, the command separator symbol, and redirection symbols, within macros. There are times, however, when it is desirable to use the dollar sign as a normal character. For example, suppose you used a dollar sign as the file name extension for files that contained financial information. The following three files might exist:

```
SALES.$
COSTS.$
BALANCE.$
```

The command DIR *.$ would list these files out. However, if this command were used as a macro definition, the $ would be ignored when the macro was executed because it would be seen as the beginning of an unrecognized special symbol. Consequently, to use a dollar sign as a regular character in a macro, you must double it. The above DIR command could be defined in a macro named DM as follows:

```
DOSKEY DM=DIR *.$$
```

The Doskey Command Syntax

Now that all of the features of Doskey have been reviewed, the complete command syntax can be examined and understood. The full DOSKEY command, with all possible switches, is:

```
DOSKEY [/REINSTALL] [/BUFFER=SIZE] [/MACROS] [/HISTORY]
  [/INSERT or OVERSTRIKE] [MACRO=text]
```

The square brackets indicate that the enclosed switch or parameter is optional.

/REINSTALL is used to install a second copy of Doskey. When this is done, the command stack and macro buffer is cleared, removing any commands that have been remembered or macros that have been defined. However, the first copy of Doskey is not removed. Instead, a second copy in installed in memory, in addition to the first one. One reason for doing this is to increase the size of the buffer. Once a copy of Doskey is installed, the buffer available to it cannot be increased, but when a second copy is installed, a larger buffer can be specified for it. However, the memory occupied by the first copy of Doskey is lost until you reboot the computer, so this should only be done when absolutely necessary.

/BUFFER=SIZE is used to specify how much memory Doskey should reserve for storing the command stack and macro definitions. The same memory area is used for both, so the more macros you have defined, the smaller the command stack will be. On the other hand, macros will take memory from the command stack as needed, so having a large number of commands in the command stack will not prevent you from defining additional macros. As you define new macros, commands will disappear off the top of the stack. Memory considerations for using Doskey are discussed in detail in the Advanced Topics section of this chapter.

/MACROS causes Doskey to list all of the currently defined macros.

/HISTORY switch lists all of the commands stored in the command stack

/INSERT and **/OVERSTRIKE** are used to specify whether Doskey should start up in the INSERT or OVERSTRIKE mode. INSERT mode causes any new text typed in the middle of a command to be inserted into the existing text, while OVERSTRIKE mode causes new text to replace existing text. If neither of these options is specified, Doskey starts in OVERSTRIKE mode. Once Doskey has been loaded, executing the DOSKEY command with one of these options sets the mode to the one specified. Alternatively, once Doskey is loaded, ins can be used to toggle between the two modes.

When Doskey is started, or the mode is changed using the DOSKEY command with either the INSERT or OVERSTRIKE mode specified, the cursor appears as a thin underscore character, regardless of the mode. When ins is used to toggle between the modes, the cursor changes into a thicker underscore character. In other words, the thin underscore character indicates the mode set by the DOSKEY command, while the thick underscore indicates the alternate mode toggled by ins.

MACRO=Text is used to create a macro. The word MACRO represents the keystrokes to be used for the macro, while the word Text represents the command line that should be stored in the macro.

Advanced Topics

There are three topics presented here:

- the use of the DIRCMD environment variable to preset switches for the DIR command
- how the Doskey program works
- managing the memory use of Doskey.

The DIRCMD Environment Variable

Since the DIR command is probably the most frequently used of all DOS commands, DOS 5.0 provides an efficient means for customizing the default switch settings. This is done by setting an environment variable named DIRCMD.

When the computer is booted, DOS reserves an area of memory known as the "DOS environment." This area is used to store various items of information that DOS routinely uses, such as the PATH, the PROMPT, and the command line template. Some of the environment space is available to store information that you specify with the SET command, which creates "environment variables." For example, the following command creates an environment variable named LIB that stores C:\WP\LIBRARY as the name of a directory

```
SET LIB=C:\WP\LIBRARY
```

Once an environment variable is created, any program can access its contents by referring to its name. In the above example, if a program was designed to always search for certain files in a directory specified by an environment variable named LIB, then they would always search the C:\WP\LIBRARY directory. If you wanted to place those files in a different directory, all you would have to do is change the SET command to place a different directory name in the LIB environment variable.

The DIRCMD environment variable is one that the DIR command always looks for when it is executed. If the DIRCMD variable exists, its contents are appended to the end of the DIR command. Consequently, by storing a directory name, a file name, or any combination of the DIR switch settings, in the DIRCMD environment variable, you can make the default DIR command execute with those settings. For example, if you wanted the DIR command to always use the /P parameter, to make it pause after each information screen, the following SET command could be used:

```
SET DIRCMD=/P
```

With this setting, whenever the DIR command is executed, DOS will append the /P switch to the command. A switch defined in the DIRCMD variable may be overridden by specifying the switch with a minus sign in front of it. For example, if the above DIRCMD were defined and you wanted to display a directory with the /P switch, you would enter:

```
DIR /-P
```

This only overrides the DIRCMD setting for a single execution of DIR; the next time DIR is entered, the /P will again be in effect.

Another use of the DIRCMD variable is to specify a default directory for DIR to use. If you always wanted DIR to display the C:\WP\LETTERS directory in wide format (the /W switch), the following command would create the appropriate environment variable:

```
SET DIRCMD=C:\WP\LETTERS /W
```

To view a different directory, specify the directory name on the DIR command line. For example, to view the root directory, enter the following command:

```
DIR \
```

Even though this overrides the directory specified in DIRCMD, it does not affect the /W setting — the root directory will be displayed in wide format, unless you also turn off the /W switch by specifying /–W.

STOP

Note: There can be no spaces around the equal sign in the SET command. This is mandatory. The environment variable will not be set properly if you leave a space before and/or after the equal sign.

Environment variables are usually set in the AUTOEXEC.BAT file. This saves you the time required to memorize them and enter them each time you start the computer. However, they can be set at the command line as well, and they can be removed from the environment. To remove an environment variable, you use the SET command but don't place a value after the equal sign. The following command will remove the DIRCMD variable from the environment, restoring the DIR command to its normal operation:

```
SET DIRCMD=
```

You may use the SET command to view the contents of the DOS environment, by entering SET by itself. This will show you the current PATH definition, the PROMPT definition, and any environment variables that have been defined.

There is a finite amount of space available in the environment. The default value is 256 bytes, or enough room to store a total of 256 characters. If you are running a lot of programs that require environment variables, or have a very large path and prompt, you may run out of environment space. When this happens, DOS displays the error message "Out of environment space." The amount of environment space can be adjusted by including the SHELL command in the CONFIG.SYS file.

When the computer is booted, the COMMAND.COM program is loaded into memory. This is the core DOS program, which manages the keyboard, screen, disk drives, and other hardware devices. COMMAND.COM also sets up the environment space. The SHELL command can be used to modify the things COMMAND.COM does when it is loaded. This is NOT the same as the DOSSHELL command, which starts the DOS 5.0 Shell that provides a graphical user interface to DOS. The DOS 5.0 Shell is the subject of Chapter 2. The use of the SHELL command is discussed in detail in Chapter 9. To adjust the size of the environment space, include this command in your CONFIG.SYS file:

```
SHELL=COMMAND.COM /E:xxx
```

In this command, `xxx` represents the number of bytes to be reserved for the environment. This number must not be less than 160, or greater than 32,768. DOS automatically rounds the number up to the nearest multiple of 16, so that specifying 170 would produce an environment space of 176 bytes. If you specify a number that does not fall in the range of 160 to 32,768, DOS will display the message "Parameter value not in allowed range," and will set the environment to its default of 256 bytes.

You are given flexibility in defining the size of the environment because any memory allocated to the environment is taken away from memory that can be used by software programs. With more and more programs requiring large amounts of memory, it is sometimes important to fine tune the environment space to use the minimum amount required.

Doskey — How it Works

Doskey is a type of program known as a TSR, or Terminate and Stay Resident program. These programs are so called because after they terminate and return control to DOS, they actually stay resident in the computer's memory, where they continue to function behind the scenes.

Some of these programs remain essentially dormant in memory until they are called up, by pressing a special combination of keys. A popular example is the Sidekick program. Sidekick provides you with a pop-up notepad, calendar, and other utilities, whenever you hold down both ⇧Shift keys at the same time. Doskey is a little different from such programs, in that it is always active, but remains behind the scenes — it is not necessary to press any special keys to activate Doskey after it is loaded. Instead, Doskey intercepts everything that is typed in at the keyboard, and interprets it before sending it on to the command line. If a Doskey macro is entered, Doskey expands the macro and translates it into the appropriate DOS commands, which are then sent to the command line. If anything else is entered, Doskey passes it along unchanged.

Doskey Memory Management

Since Doskey remains in memory while other programs run, it reduces the amount of memory available to those programs. While some TSRs use a lot of memory, Doskey is relatively small — with its default command stack and macro buffer of 512 bytes, it occupies about 4K of memory.

When Doskey is loaded, the entire buffer is available for the command stack. As macros are defined, they are stored in the buffer, reducing the amount of memory available to the command stack. However, the command stack will never take memory away from macros. If the command stack fills the buffer, and a macro is defined, the commands at the top of the stack are eliminated to make room for the macro. If the buffer is filled, with macros, the command stack will not store any commands.

The buffer size can be specified at the time of loading Doskey, with the /BUFSIZE=xxx switch, where `xxx` represents the number of bytes to reserve for the buffer. If /BUFSIZE is not specified, the buffer is 512 bytes. The minimum value of xxx is 256. Since the buffer cannot be increased once Doskey is loaded, the only way to provide additional memory is to load a second copy of Doskey, using the /REINSTALL switch. This wastes memory, since the first copy of Doskey is still installed but no longer used.

You can free up memory for additional macros by deleting ones that already exist. However, once a part of the buffer has been allocated to a macro, it cannot be released for use with the command stack — it can only be released for storing another macro. It is important, therefore, to plan your use of the command stack and macros carefully, allocating a buffer large enough to hold both, but not so large that some of it is not used.

Chapter 4

Understanding and Managing Memory

While disk files are used for permanent storage of information — both software programs and data — information must be transferred from the disk to electronic memory before it can be used by the computer. The CPU (Central Processor Unit) then accesses the information from memory as it is needed. There are no physical laws that require this. You could design a computer in which there was no electronic memory, where all information was transferred directly between the CPU and disk files. However, reading and writing to disks is agonizingly slow in comparison to reading and writing to electronic memory. A computer designed to use disk files instead of electronic memory would be useless for today's applications because it would take too long to do anything.

When the computer needs to run a program, it reads the entire program into memory at once. This is referred to as "loading" the program. After the program is loaded, the CPU starts executing the instructions in the program, reading them one at a time from their memory locations. When the program needs to access data — say from a file to be edited in a word processor — the computer copies that data into memory. The CPU then retrieves the data from memory as it is needed. Consequently, the computer must have enough memory to store both the program and the data it uses.

The memory capacity of Personal Computers has grown tremendously since PCs first appeared. The original PCs were designed to have a maximum of 640K of memory, which at that time seemed like more than anyone would ever need on a PC. Personal Computers were typically purchased with 256K installed, and more memory was added if and when it was needed. As software evolved, PCs were used for more and more complex tasks — tasks that previously had been tackled only by expensive mainframe computers. To manage these tasks efficiently, more and more memory was required on PCs. It became increasingly difficult, and in some cases impossible, to fit all of the program code and data into 640K of memory.

Today, machines with two megabytes of memory are common, and it is possible to purchase Personal Computers that have over 100 megabytes of memory. However, because the original design of PCs embodied the philosophy that 640K was more memory than anyone would ever need, the operating system for PCs was designed to only use 640K. Since the machines themselves could only manage this much memory, there was no point in writing an operating system that could handle more. No one anticipated that the PC would rapidly evolve into a machine that could handle over a thousand times that much memory. This limitation of 640K of memory is so deeply embedded in the design of DOS that it is almost impossible to change. Consequently, making use of memory above the 640K limit is not straightforward. There are many ways in which memory can be configured, depending on what type of PC you have, how much memory the PC has, and how the memory is installed in the PC. This chapter discusses all of these aspects of computer memory, as well as the DOS 5.0 commands that can be used to control how memory is configured, and how it is used by the software.

Throughout this chapter, mention is made of the different types of CPU chips. The term 8088/8086 refers to all Personal Computers that use either the 8088 or 8086 CPU chip. The terms 286 and 80286 are used to refer to PCs that use the 80286 chip; and the term 386 and 80386 are used to refer to PCs that use the 80386 or 80486 chip.

Understanding Memory Resources

Physically, there are two kinds of memory in PCs: RAM and ROM. RAM stands for **R**andom **A**ccess **M**emory. RAM is so called because it can be accessed in a random sequence — the computer can instantly retrieve information that is stored anywhere in RAM, without having first to retrieve everything that appears ahead of the desired information. This is the memory that is usually referred to when just the term "memory" is used. For example, a machine that is said to have "640K of memory" has 640K of RAM.

For the purposes of this chapter, the terms "RAM" and "memory" will be used interchangeably. Where it is necessary to mention ROM, it will be referred to explicitly.

ROM stands for **R**ead **O**nly **M**emory. ROM also allows random access, but as its name implies, the computer cannot write information into ROM, information can only be retrieved. ROM contains information that should never be changed and this information is built in to the ROM before it is installed in the computer. For example, there are some fundamental routines the computer uses for accessing peripheral devices, such as disk drives and printers. These routines are known as the **B**asic **I**nput **O**utput **S**ystem or BIOS. These routines are also referred to as ROM-BIOS. When DOS needs to access a device, it sends a request to the BIOS, which then handles the actual communication with the device. Since the BIOS handles the very low-level communications between the computer and its devices, the BIOS routines should never be changed. The devices themselves are designed to communicate with the BIOS in a particular way, so if the BIOS were changed, the various parts of the PC may not be able to function together. Consequently, the BIOS is written into ROM.

There are two advantages to having something stored in ROM: ROM is non-volatile memory, which means that it does not lose its contents when the computer is turned off. Consequently, information stored in ROM is always available as soon as the computer is booted; it does not have to first be read in from a disk. In fact, the initial instructions that tell the computer what to do when you boot it are stored in ROM. The second advantage to having something in ROM is that it cannot be inadvertently changed by a program. The only way to change the contents of ROM is to replace the ROM chips with ones that contain different information.

Some ROM is installed on the system board when you buy your computer, and some may be on add-on boards that are installed in the computer's expansion slots. The BIOS routines are in ROM that is installed on the system board.

When you buy a Personal Computer, it comes with a certain amount of RAM installed. You may add more later, up to some maximum amount that the computer is capable of managing. This maximum amount varies from one manufacturer and model to another. There are two places on the computer where RAM can be stored: on the system board (also known as the mother board); and on expansion cards that plug into expansion slots on the system board. Most computers are capable of using more memory than will fit on their system board and when the system board is filled to capacity, it is necessary to install expansion boards to add more memory.

Bits and Bytes

In order to talk about memory, it is necessary to know some terms that are used to describe it. Electronic memory consists of millions of tiny switches, which can either be on or off. These switches are referred to as "bits" and a bit is, therefore, the smallest amount of information a computer can store. A bit can be set to "on," which is interpreted by the computer as a one, or "off," which is interpreted by the computer as a zero. Since a single bit can only mean one of two things, in order to represent complex information, computers work with groups of eight bits; a group of eight bits is called a "byte." Software that runs on IBM-compatible Personal Computers most often works with one byte at a time, but some software uses groups of two or four bytes. These groups are known as "words;" a two-byte word is called a 16-bit word, and a four-byte word is called a 32-bit word.

It takes a large number of bytes for the computer to be able to do anything useful, so memory is measured in terms of kilobytes and megabytes. A kilobyte is 1024 bytes; a megabyte is 1,048,576 bytes. The term "kilobyte" literally means 1000 bytes, but the computer does everything in multiples of two, since it is operates on the base two number system. 1024 is two raised to the tenth power (2^{10}); since it is approximately 1000, 1024 bytes is referred to as a kilobyte. Similar reasoning leads to the term megabyte, which is approximately one million bytes. Thus, when you hear of a computer having 640K, this means the computer has 640 X 1024, or 655,360 bytes of memory.

How does a computer manage to perform all the complex tasks it does, if everything it works with consists only of a series of 0s and 1s? The computer distinguishes between two different interpretations of the contents of a byte: one interpretation is as an instruction to the CPU; the other is as data that the CPU can work with. When a byte contains part of a program the computer is running, the CPU interprets the contents of the byte as containing an instruction. There are several hundred instructions the CPU understands, and all of the computer's actions ultimately are reduced to sequences of these instructions. An example of an instruction that the CPU understands is:

```
ADD AL 13
```

This instruction adds 13 to the value stored in a part of the CPU known as the AL register. The number of bytes required to store an instruction varies with the instruction. Internally, the ADD AL 13 instruction would occupy three bytes; the 13 would be represented as 00001101 — the binary version of the number 13. The ADD and AL would be represented by binary codes that are interpreted by the CPU as ADD to the AL register.

When a byte contains data, rather than an instruction for the CPU, then the computer interprets the byte as having a numeric or text meaning. For example, the binary number 00101101 could represent an instruction to the CPU that tells it perform some operation on a number, or it could represent the number 45.

By having two different ways of interpreting the same binary codes, the computer can store instructions to the CPU in the same memory as it stores data. By differentiating between the two, and accurately interpreting the meaning of each byte in its memory, the computer carries out all of the various tasks that it performs.

How Memory is Organized

Since 640K of memory is 655,360 bytes, and a byte is the basic unit of information storage, there are 655,360 places in which an item of information might be found in a 640K PC. In order to keep track of all these locations, and to make it easy to access any particular location quickly, the computer assigns an address to each byte, starting with 0 and proceeding up to the last byte of memory the computer contains.

The limiting factor in determining how much memory a computer can have is the size of the address. Computers use binary addresses — numbers consisting only of ones and zeros. DOS is designed to use addresses that are 20 bits long, so the largest address it can reference is 11111111111111111111, which is 1,048,575 in decimal notation. Since the first address is 0 and the last address is 1,048,575, DOS can reference up to 1,048,576 bytes of information — exactly one megabyte.

While DOS is capable of addressing up to one megabyte of memory, only the first 640K of that memory is available for programs and data. This is because the PC reserves the memory addresses from 640K to 1 megabyte for special purposes. The memory area from zero to 640K is termed "conventional memory," because it is the area that is conventionally used for running programs. The memory from 640K to 1 meg is referred to by several names, including reserved memory, upper memory area, or high memory. The region from 640K to 1 megabyte contains the addresses for 384K of memory. To see where the number 384 comes from, consider what the terms "640K" and "1 megabyte" mean: 640K is 640 X 1024, or 655,360 bytes, since there are actually 1024 bytes in 1K. Likewise, 1 megabyte is 1024 X 1024, or 1,048,576 bytes. Subtracting 640K from 1 megabyte leaves 393,216 bytes left in the upper memory area. 393,216 divided by 1024 is 384; thus, there are 384K of memory addresses in the upper memory area.

There is a variety of ways in which the computer makes use of the upper memory area. For example, images displayed on the screen must be stored in memory, so a part of this upper memory area is reserved for the contents of the screen. The area of memory reserved for the screen is also known as video memory. The computer is constantly monitoring the video memory, and updating the screen whenever the contents of video memory change. Read Only Memory (ROM) was mentioned earlier in this chapter; all ROM addresses are in the region from 640K to 1 meg. The ROM BIOS routines are stored in addresses at the very top of this region. If a network adapter card is installed in your computer, there is ROM on that card whose addresses appear in the upper memory area. Any other expansion cards that contain ROM will use addresses in the upper memory area as well.

A typical configuration of the memory in a 640K Personal Computer is shown in Figure 4-1.

ADDRESS RANGE		Approx.	
Hexadecimal	Decimal	Size	Description
00000 - 003FF	0000 - 1023	1K	Interrupt Area
00400 - 004FF	1024 - 1279	0.3K	BIOS Data Area
00500 - 006FF	1280 - 1791	0.5K	System Data
00700 - 0DFBF	1792 - 57279	54K	DOS
0DFC0 - 0F28F	57280 - 62095	4.7K	Command.com
0F290 - 9FBFF	62096 - 654335	578K	Available for programs
9FC00 - 9FFFF	654336 - 655359	1K	UMB
A0000 - AFFFF	655360 - 720895	64K	VGA Graphics
B0000 - B7FFF	720896 - 753663	32K	UMB
B8000 - BFFFF	753664 - 786431	32K	VGA Text
C0000 - C7FFF	786432 - 819199	32K	Video ROM
C8000 - EFFFF	819200 - 983039	160K	UMB
F0000 - FFFFF	983040 - 1048575	64K	System ROM

Figure 4-1 — Memory Layout of a 640K Personal Computer.

As you can see in Figure 4-1, a 640K computer actually has more than 640K of memory. The 640K refers to the RAM available for running programs — the RAM from address 0 to address 655,359. How much more memory the computer has depends on what additional devices are attached to it. A monochrome text video adapter will not have as much video memory as a color graphics adapter. Some PCs will have adapter cards, such as those used for connecting to local area networks, that have ROM on them.

If all of the reserved memory were filled — either with reserved RAM, such as the video memory contains, or with ROM — then a 640K PC would actually have 1 meg of memory. However, this is never the case. There are always gaps in the reserved memory area. Where those gaps lie, and how big they are, varies from one PC to the next, depending on the exact hardware devices the computer has. These gaps are known as "Upper Memory Blocks," or UMBs for short. Several UMBs are shown in Figure 4-1.

If the first meg of memory is accounted for on a computer that is said to have 640K of memory, then a question arises: how much memory does a computer have if it is referred to as having 1 megabyte of memory? The answer is that it has exactly what a 640K computer has, plus an additional 384K. However, the addresses for this additional 384K lie above the one megabyte address, and therefore cannot be referenced directly by DOS, since the largest address DOS can store is 1 megabyte. This 384K of additional memory is called "extended" memory, since it extends the addresses of the computer beyond the traditional 1 megabyte limit. A computer that is said to have two megabytes of memory would have 1408K of extended memory: 384K, plus 1024K (which is one megabyte), equals 1408K. Extended memory is discussed in greater detail later in this chapter.

How DOS Uses the First 640K of Memory

DOS itself consists of two parts: an "internal" part and an "external" part. The internal part is in a program named COMMAND.COM, which is loaded into memory when the computer is booted. All of the internal DOS commands are stored in COMMAND.COM. The external part is a set of programs that are run from the disk, just like any other program. The Doskey program is an example of the external part of DOS. The Command Dictionary (Chapter 10) indicates for each command whether it is stored internally or externally.

Since COMMAND.COM is loaded permanently into memory when the computer is booted, it uses up some of the first 640K of memory. The size of COMMAND.COM grew with each new release of DOS through version 4.0; thus, each new version of DOS left less memory for programs. In DOS 5.0, the size of COMMAND.COM has been reduced to occupy 64K of memory; this is less than was required in DOS 4.0, but still more than DOS 3.3 used.

In addition to loading COMMAND.COM into memory, DOS sets up an area of memory for the DOS environment. The DOS environment contains information that DOS needs to have as it goes about directing the computer's operation. For example, the current Path, as defined by the PATH command, is stored in the environment, as is the current PROMPT definition. If there are environment variables defined, these are also stored in the DOS environment. The amount of memory reserved for the environment is 256 bytes by default, but this can be varied using the SHELL command, as discussed in Chapter 3.

All programs and all data are stored in files. For the computer to run a program or access the data in a file, DOS must first "open" that file. For example, when a word processor is used, the word processor program is stored in one or more files, and each of these must be opened before DOS can load the program. Each time you edit a document in a word processor, the file that stores the document must also be opened. Many word processors allow you to have many document files open at once. When you execute one of the DOS external commands, such as DOSSHELL, the command is a program that is stored in a file. Database programs typically require numerous files to be open in order to manage all the data in the database.

Opening a file requires that DOS identify the file name, locate it on the disk, load it into memory, and remember where it is stored on the disk. A number, called a "file handle," is associated with each open file. When a program opens a file, DOS assigns a file handle to it and tells the program what that number is. The application then refers to the file handle, rather than the file name itself.

For example, when you want to edit a file called "MYFILE.LET" with a word processor, you first identify the file name. The word processor then passes that name to DOS, where the file handle is assigned and passed back to the word processor. If DOS assigns the file handle 8 to the file, the word processor then refers to "file 8" when it reads or writes that file.

Memory is required to keep track of the file handles. This memory must be allocated when the computer is booted; DOS cannot increase or decrease the amount of memory available for file handles after the computer is in use. Allocating memory for file handles is done with the FILES command in the CONFIG.SYS file. The following command specifies that DOS should reserve memory for having up to 16 files open at the same time:

```
FILES=16
```

The maximum number of files that may be specified with the FILES command is 255. If there is no FILES command in the CONFIG.SYS file, the default value of 8 is used and 8 is the minimum number that may be specified. DOS requires 64 bytes of memory for each file handle, so if 18 file handles are specified in the CONFIG.SYS file, DOS will reserve 1152 bytes of memory for them. For some applications, the default value of 8 file handles is sufficient. However, if you ever see an error message that indicates there are not enough file handles available, you must increase the number of files specified in CONFIG.SYS.

Applications that require more than the minimum number of file handles usually point this out in their installation documentation. Some programs will automatically adjust the CONFIG.SYS file when they are installed, if the number of file handles is too low. Note that the computer must be rebooted for any changes to the CONFIG.SYS file to be come effective.

Another command that appears in the CONFIG.SYS file, and that affects available memory, is the BUFFERS command. BUFFERS sets up memory buffers that increase the efficiency with which data can be transferred between disks and memory. The BUFFERS command is discussed in Chapter 6, but for now it is important to understand that each Buffer occupies 532 bytes of memory. If your CONFIG.SYS file contains a BUFFERS=20 command, then 10,640 bytes of memory will be taken for Buffers.

The CONFIG.SYS file may also contain one or more DEVICE commands. These commands take the following form:

```
    DEVICE=filename
```

Where **filename** is the name of a file that contains the device driver. For example, if you use a mouse with your computer, then a device driver for the mouse is loaded, communicating between the mouse and the computer. Device drivers are discussed in detail in Chapter 9. The DEVICE command loads the device into memory, where it stays until the computer is turned off. The amount of memory taken up by device drivers varies widely, from less than a hundred bytes to thousands.

Besides COMMAND.COM, file handles, buffers, and device drivers, there are programs known as TSRs (**T**erminate and **S**tay **R**esident). A TSR program remains in memory after it returns control to DOS. From memory, it can be recalled by pressing a special key combination. A program named "Sidekick" is probably the most well-known TSR. After Sidekick is loaded into memory and control is returned to DOS, Sidekick can be recalled by pressing both Shift keys at the same time. Sidekick then provides you with a note pad, a calculator, a phone dialer, and other utilities. Pressing both Shift keys again makes Sidekick disappear until you call it up again. The Doskey program, discussed in Chapter 3, is another example of a TSR.

TSRs are very useful — they provide handy utility and time-saving features that can be used while you are using the computer for your work. However, they require memory. Programs like Sidekick can occupy anywhere from 30K or 40K to over 100K, depending on the particular configuration you choose. Doskey takes up about 5K.

There are, then, the following five different ways in which DOS uses the first 640K of memory, each of which reduces the amount of memory available for programs:

- COMMAND.COM
- File handles
- Disk buffers
- Device Drivers
- Terminate and Stay Resident Programs.

On some computers, after all of the required configuration commands and device drivers are loaded, there is less than 450K of available memory left to run programs. TSRs can reduce this even further. There are several features built in to DOS 5.0 to help reduce the amount of memory used by these resources; these features are discussed later in this chapter.

Expanded Memory

As Personal Computers evolved, it became evident that some means would have to be found for getting past the 640K barrier. Software was appearing that, in some cases, was capable of performing tasks that were more complex than could be accomplished in the available memory. In other cases, the lack of memory slowed down the processing to unacceptable rates because disk space had to be used in lieu of additional memory.

In understanding how this problem was addressed, it is important to keep in mind that there are two aspects to the 640K barrier: the first is the 20-bit addressing limitation of the 8088 chip — the CPU used in the first IBM Personal Computers. This is a hardware problem and there is nothing that can be done that will allow that chip to directly address more than one megabyte of memory. The second aspect of the problem is the addressing capability of DOS. This is a software problem; no matter how much memory the hardware in a particular PC is capable of addressing, as long as that PC is running DOS it will only be able to directly address one megabyte of memory. CPUs that were developed after the 8088 chip — the 80286, 80386, and so on — are capable of addressing more memory than 8088. The 80286 can address up to 16 megabytes of memory, and the 80386 and subsequent chips can address up to four gigabytes of memory. They do this by having more than the 20 address bits available to the 8088 chip. However, since DOS uses the same 20-bit address as the 8088, these chips were designed to run in different "modes." When they are running on PCs that use DOS, they run in what is called "Real mode." Real mode makes them emulate the 8088 chip, dropping their address size down to 20 bits and limiting them to one megabyte of memory. To access memory beyond the one megabyte barrier, these chips use a mode called "Protected mode."

There were, then, really two problems to address: the first was the hardware limitation of PCs running in Real mode — whether they were 8088-based machines or had a more advanced chip in them; and the second was how to make the greater addressing capability of the later chips available to DOS. The hardware problem was dealt with first, by developing a technology known as "expanded memory." The term expanded memory was used because the technology effectively expanded the existing memory, without requiring either the hardware or DOS to use any new addresses.

Expanded memory technology was developed as a joint venture between three companies: Lotus, Intel, and Microsoft, and is therefore known as LIM EMS, for **L**otus **I**ntel **M**icrosoft **E**xpanded **M**emory **S**pecification. The first version was dubbed Version 3.2, the recent version is 4.0. From a technical perspective, implementing EMS is quite complex. However, conceptually it is not too difficult to understand.

Expanded memory technology uses a technique known as "mapping." Mapping allows the computer to use a large amount of memory by examining it in small pieces. To visualize this, imagine that you have a very large sheet of paper, covered with numbers, but you are only able to look at this sheet through a small, immovable frame, which shows only a tiny part of the sheet. Now imagine that you can cut the sheet of paper, into pieces just the size of your frame, and can put whatever piece of paper you want behind the frame. It is now possible to view the entire sheet of paper, albeit one small frame at a time. This process, of moving different parts of the sheet of paper behind the frame, is referred to as mapping.

On the computer, mapping is accomplished by setting up a 64K region of memory known as the "page frame." The page frame lies somewhere within the first megabyte of memory, so it can be read from and written to by DOS. The page frame itself is divided into four 16K pages. Special memory management software is run that creates the page frame and manages its use.

An expanded memory board is then added to the computer, which can contain up to 32 megabytes of memory (under LIM EMS Version 4.0). This board must be specifically designed to have its memory used as expanded memory. This memory is not addressable by normal DOS. Instead, it is addressed using a technique that is built in to the memory management software that created the page frame. Now, when a program wants to use memory beyond one megabyte, it uses the address of the page frame, which DOS recognizes. The program then informs the memory management software what part of the expanded memory it wants to use. The memory management software then maps that area of EMS into one of the pages in the page frame. If the area of EMS that is wanted is larger than a single page, multiple pages are used; if it is larger than the entire page frame, it is broken into segments that are mapped in one at a time. Figure 4-2 shows a 64K page frame mapped into memory, beginning at address 917,506.

ADDRESS RANGE		Approx.	
Hexadecimal	Decimal	Size	Description
00000 - 003FF	0000 - 1023	1K	Interrupt Area
00400 - 004FF	1024 - 1279	0.3K	BIOS Data Area
00500 - 006FF	1280 - 1791	0.5K	System Data
00700 - 0DFBF	1792 - 57279	54K	DOS
0DFC0 - 0F28F	57280 - 62095	4.7K	Command.com
0F290 - 9FBFF	62096 - 654335	578K	Available for programs
9FC00 - 9FFFF	654336 - 655359	1K	UMB
A0000 - AFFFF	655360 - 720895	64K	VGA Graphics
B0000 - B7FFF	720896 - 753663	32K	UMB
B8000 - BFFFF	753664 - 786431	32K	VGA Text
C0000 - C7FFF	786432 - 819199	32K	Video ROM
C8000 - DFFFF	819200 - 917503	96K	UMB
E0000 - EFFFF	917506 - 983039	64K	Page Frame
F0000 - FFFFF	983040 - 1048575	64K	System ROM

Figure 4-2 — Memory Layout Showing EMS Page Frame.

The advantage of using EMS memory is that it provides more memory than DOS allows, without interfering in any way with DOS or with programs written to run under DOS. On the other hand, in order to use EMS memory, a program must be written specifically to take advantage of it, since the program must make calls to the expanded memory manager. Furthermore, it is slower to use EMS memory than conventional memory because there is an extra step involved — calling the expanded memory manager — and because the memory must be mapped back and forth between the page frame and EMS memory in relatively small chunks.

Computers built around the 80386 or later CPU do not require a special expanded memory board to use expanded memory — all they require is memory management software that can manage the page frame and EMS. This is because the expanded memory hardware capabilities are built into the chips.

DOS 5.0 includes an expanded memory manager, EMM386.EXE, which can be used to convert extended memory to expanded memory on 386 machines. 8088 and 286 PCs that have expanded memory boards must use the expanded memory manager that came with the board, because each board manufacturer provides their own unique expanded memory manager.

Extended Memory

Extended memory is simply memory that has addresses above one megabyte. A computer that has an 80286 or later chip, and that is described as having more than 640K of RAM, has extended memory. Thus, as described above, a computer that is said to have one megabyte of RAM has 640K of conventional RAM and 384K of extended RAM. However, because of the one megabyte address limitation of DOS, this memory is non-existent as far as DOS is concerned. Extended memory cannot exist on a computer based on the 8088 chip, since that chip itself can only recognize one megabyte of memory.

If your CPU can directly address its extended memory, then the only problem that needs to be overcome is that DOS is unable to address it. There are three ways of doing this.

The first way is to purchase software that is designed to use it directly. Such software uses a technology known as a "DOS Extender," which shifts the CPU into protected mode, where it can directly address extended memory. The DOS extender then interfaces between DOS and the software, essentially making DOS capable of using extended memory for the particular program.

DOS extenders are costly for software developers and can create conflicts when they are used in conjunction with other programs. Consequently, four companies — Lotus, Intel, Microsoft, and AST — jointly developed a second solution to the extended memory problem — the Extended Memory Specification (XMS). DOS 5.0 has features that are compatible with version 2.0 of XMS. XMS provides a standard technique for software programs to use when accessing extended memory, without employing a DOS extender. In order for the programs to do this, XMS memory manager software must be installed. DOS 5.0 includes the device driver to manage XMS compatible software.

The third way to use extended memory is to convert it to expanded memory, then run software that can make use of expanded memory. This is currently the most common use of extended memory, since much more software has been developed to use expanded memory than extended. However, this can only be done on PCs using the 386 and later chips, because these chips have the hardware requirements for EMS 4.0 built in to them. On 286 machines, expanded memory must be installed on a special expanded memory board that requires its own unique expanded memory manager. DOS 5.0 provides a program named EMM386.EXE, which can be used to convert extended memory to expanded on 386 machines.

Windows (versions 3.0 and later) has memory managers built in that automatically take advantage of whatever type of memory and CPU chip are available on your computer. If you have an 80386 PC with extended memory, then that memory will be used in the best possible way for the software you are running under windows. If you have an 80286 PC with expanded memory, then that memory will be used as expanded memory.

Note: The versions of HIMEM.SYS and EMM386.EXE that come with DOS 5.0 are more recent than those that shipped with Windows 3.0. If you are running Windows 3.0 and DOS 5.0, you should replace the Windows 3.0 versions of these files with the DOS 5.0 versions.

Expanded memory is a short-term solution to a long-term problem. As DOS extenders become more commonplace, and as more users migrate to Windows, extended memory will become the standard and the use of expanded memory will probably disappear. Eventually DOS itself may evolve into an operating system that can make direct use of extended memory by using larger addresses. Consequently, when planning for the long term, equipment should be purchased with extended memory. If, on the other hand, you have an older or lower-end machine and all you want to do is extend the life of that machine, expanded memory is the cheapest way to do this.

Managing Memory Resources

As seen in the preceding section, there are three types of memory resources that may be available on a computer: conventional memory, which is all memory up to one megabyte; extended memory, which is memory beyond one megabyte on an 80286 or later PC; and expanded memory, which provides additional memory that does not fit into the normal addressing scheme of DOS and the 80x86 family. DOS 5.0 provides features for maximizing the use of all three types of memory.

Maximizing the Use of Conventional Memory

Figure 4-3 shows the layout of the first megabyte of memory in a typical PC.

ADDRESS RANGE		Approx.	
Hexadecimal	Decimal	Size	Description
00000 - 003FF	0000 - 1023	1K	Interrupt Area
00400 - 004FF	1024 - 1279	0.3K	BIOS Data Area
00500 - 006FF	1280 - 1791	0.5K	System Data
00700 - 0DFBF	1792 - 57279	54K	DOS
0DFC0 - 0F28F	57280 - 62095	4.7K	Command.com
0F290 - 0F500	62096 - 62720	0.6K	File handles
0F501 - 11D01	62721 - 72961	10K	Buffers
11D02 - 12D62	72962 - 77154	4.1K	Ansi.sys device driver
12D63 - 13D63	77155 - 81251	4K	Doskey
13D64 - 9FBFF	81252 - 654335	560K	Available for programs
9FC00 - 9FFFF	654336 - 655359	1K	UMB
A0000 - AFFFF	655360 - 720895	64K	VGA Graphics
B0000 - B7FFF	720896 - 753663	32K	UMB
B8000 - BFFFF	753664 - 786431	32K	VGA Text
C0000 - C7FFF	786432 - 819199	32K	Video ROM
C8000 - DFFFF	819200 - 917503	96K	UMB
E0000 - EFFFF	917506 - 983039	64K	Page Frame
F0000 - FFFFF	983040 - 1048575	64K	System ROM

Figure 4-3 — Memory Layout of a 640K Personal Computer.

About 79K of memory is occupied by DOS, file handles, buffers, the ANSI.SYS device driver, and the Doskey TSR program. If yours is a standard 640K machine, then that leaves about 560K for user memory and there is nothing more you can do about it. However, if your PC has one megabyte or more of memory, then there is at least 384K of extended memory. Looking back at Figure 4-3, there is about 129K of unused memory address space in the UMBs. Since the UMBs represent only empty ROM addresses (there are no memory chips in the sockets, and on some machines, no sockets), it is possible to "back-fill" your extended memory into this area. On PCs using 386 chips, DOS 5.0 has the capability of backfilling the UMBs with parts of this 384K, thereby making the UMBs available for normal use. In the example shown in Figure 4-3, there is a 96K UMB beginning at address 819,200, and 384K of extended memory beginning at address 1,048,576. DOS 5.0 can change the address for the top 96K of the extended memory to be 819,200. Now, if DOS or another program references address 819,200, the physical memory that was addressed at the top of the extended area would be used. This reduces the amount of extended memory by 96K, but conventional memory is generally more useful than extended memory.

On machines using 8088/8086 and 80286 chips, the DOS 5.0 memory managers cannot be used to back-fill UMBs, so a separate memory manager must be purchased in order to make use of this technique. The following examples assume that you have an 80386 PC; if you do not, the memory manager you purchase for managing the UMBs should provide instructions for making use of them.

Any item that occupies memory — whether it is the file handle memory space, a device driver, a program, or some other item — requires contiguous memory. Contiguous memory has an unbroken sequence of addresses. Thus, three stretches of memory, each of which is 80K large, can only be used individually for programs that require 80K or less; they cannot be used collectively for a program that requires 240K. This is why, in Figure 4-3, all of the memory that is occupied by DOS, file handles, and so on, is at the bottom of the memory. By putting each new item at the bottom of available memory, all memory above the last item loaded is left unbroken, thereby leaving the largest amount of contiguous memory possible for running programs and storing data.

UMBs are relatively small, and they are scattered around the upper memory area. Consequently, they cannot be used to directly increase the memory available for large programs and data files. However, items that tend to reduce the available memory — like file handles and TSR programs — tend to be small, so they are candidates for relocation into UMBs.

In order to back-fill UMBs, two things must be accomplished: the memory above one megabyte must be made available as XMS 2.0 memory (there must be a memory manager that can access this memory according to the XMS Version 2.0 specification); and a memory manager must be installed that can back-fill UMBs with extended memory. There are third party memory managers, such as QEMM-386, that can accomplish both of these tasks. In DOS 5.0, two separate memory managers are used: HIMEM.SYS and EMM386.EXE. To set up backfilling, the following two commands must be added to the CONFIG.SYS file:

```
DEVICE=C:\DOS\HIMEM.SYS

DEVICE=C:\DOS\EMM386.EXE NOEMS
```

There is a variety of ways in which EMM386.EXE can be used, backfilling UMBs is only one of them. The NOEMS switch performs two functions: it tells EMM386.EXE to back-fill UMB memory; and it tells EMM386 not to convert extended memory to expanded. This provides the maximum amount of memory possible in UMBs. When EMM386.EXE is used to convert extended memory to expanded memory, part of the UMB memory is used for a page frame. Setting up EMM386.EXE to convert extended memory to expanded memory is discussed in the next section, "Managing Expanded Memory."

Once memory is backfilled, DOS must be told to include the upper memory area when it examines memory. This is done with a DOS=UMB command in the CONFIG.SYS file. DOS=UMB tells DOS that items it normally looks for in low memory may be in high memory instead.

DOS does not automatically use UMBs for whatever it can, even after they have been set up. Therefore, the last step in using UMBs is telling DOS what to put in them, and what to leave in the usual, low memory area. This is done with the DEVICEHIGH and LOADHIGH commands.

DEVICEHIGH loads device drivers into UMBs. It is used in exactly the same way as the DEVICE command, which loads device drivers into low memory. The following command loads a driver named ANSI.SYS into a UMB:

```
DEVICEHIGH=ANSI.SYS
```

There is one variation on the DEVICEHIGH command that does not apply to the DEVICE command: the "size" parameter. Since the number and size of UMBs varies with different hardware configurations, there is no way for DOS to predict how many UMBs there will be, or how large they will be. Therefore, there is the possibility that there will not be a UMB large enough for any particular device driver. If you attempt to load a device driver using DEVICEHIGH, and there is no UMB large enough for it, one of two things will happen: DEVICEHIGH will act like DEVICE, and load the driver in low memory; or your PC may lock up, forcing you to reboot. The reason the PC may lock up is that DEVICEHIGH may believe the device driver will require less memory than it does. Therefore, DEVICEHIGH goes ahead and tries to load the driver into a UMB too small for it. This can cause the computer to lock up. Obviously, loading the driver low is preferable to locking up the computer, but given the tremendous variety of device drivers in existence, it is not possible to insure that DEVICEHIGH will always accurately perceive the memory requirements of a particular driver.

Since the DEVICEHIGH command is in the CONFIG.SYS file, if it locks up the computer there is no way to boot from the disk that contains the CONFIG.SYS file. Usually this is a hard disk, which cannot easily be replaced. Therefore, it is important to make sure you have a floppy disk from which you can boot your PC. After booting with the floppy disk, you can edit the CONFIG.SYS file on the hard disk to remove the DEVICEHIGH command. A bootable floppy disk may be created simply by putting a floppy disk in a drive and entering the command FORMAT A:/S (assuming that the floppy drive is the A drive).

To avoid the problem of DEVICEHIGH locking up the PC, DEVICEHIGH can accept a parameter, telling it the minimum amount of space that must be available in a UMB for it to load the driver in high memory. This parameter uses hexadecimal (hex), or base 16, numbers. The following command tells DEVICEHIGH to load the ANSI.SYS driver into a UMB, if there is a UMB with at least 4192 bytes free (the decimal number 4192 is represented as 1060 in hex):

```
DEVICEHIGH SIZE=1060 ANSI.SYS
```

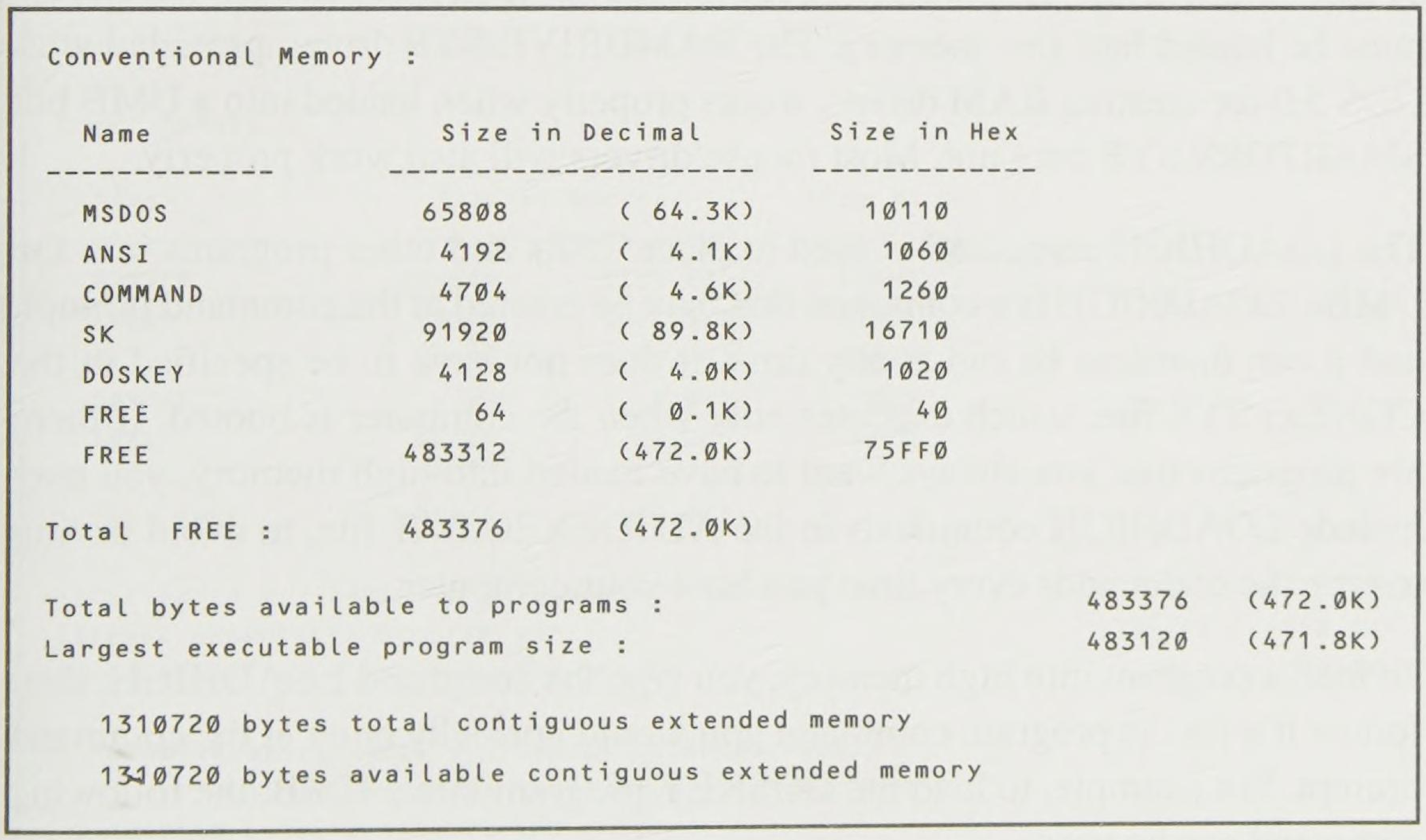
To determine the number of bytes required by a device driver, the driver should be loaded into low memory with a DEVICE command, then the following command entered at the command prompt:

```
MEM /CLASSIFY | MORE
```

This command displays a list of the items currently in memory. For each item in memory, the following information is shown:

- the name of the item
- the item's size in bytes, in decimal format, followed by the number of Kilobytes in decimal format
- the item's size in hexadecimal format.

Figure 4-4 shows a sample screen of this output. Note that the ANSI.SYS driver is designated as requiring 4192 bytes, or 1060 bytes in hexadecimal format.



```
Conventional Memory :

  Name                 Size in Decimal       Size in Hex
-------------      ---------------------    -------------
  MSDOS               65808      ( 64.3K)      10110
  ANSI                 4192      (  4.1K)       1060
  COMMAND              4704      (  4.6K)       1260
  SK                  91920      ( 89.8K)      16710
  DOSKEY               4128      (  4.0K)       1020
  FREE                   64      (  0.1K)         40
  FREE               483312      (472.0K)      75FF0

Total  FREE :        483376      (472.0K)

Total bytes available to programs :                          483376   (472.0K)
Largest executable program size :                            483120   (471.8K)

   1310720 bytes total contiguous extended memory
   1310720 bytes available contiguous extended memory
```

Figure 4-4 — MEM Command Output.

After determining the size of the device driver with the MEM command, use this value with the SIZE parameter on the DEVICEHIGH command. For example, to specify the size for ANSI.SYS, the following DEVICEHIGH command is used:

```
DEVICEHIGH SIZE=1060
```

This technique will work for most device drivers. However, some drivers use extra memory during the loading process, then cut back to their final size. In this case, the size shown with the MEM command will not provide enough memory to load the driver, resulting in the either the driver being loaded into low memory or the PC locking up. In this case, you must either find out from the manufacturer of the driver how much memory is required to load the driver, or incrementally increase the size allotment until DEVICEHIGH loads the driver into high memory.

Most device drivers will run successfully in a UMB, if there is a large enough UMB available. However, some are not designed to correctly use the addresses in the upper memory area, in which case they will not function when loaded high, and may cause the PC to lock up. Since HIMEM.SYS and EMM386.EXE must be loaded before the UMBs can be used to load devices, these two device drivers must be loaded into low memory. The RAMDRIVE.SYS driver, provided with DOS 5.0 for creating RAM drives, works properly when loaded into a UMB but SMARTDRV.SYS does not. Most mouse drivers will also work properly.

The LOADHIGH command is used to place TSRs and other programs into the UMBs. LOADHIGH is a command that may be entered at the command prompt, and it can therefore be run at any time. It does not have to be specified in the CONFIG.SYS file, which executes only when the computer is booted. If there are programs that you always want to have loaded into high memory, you may include LOADHIGH commands in the AUTOEXEC.BAT file, to avoid having to type the commands every time you boot your computer.

To load a program into high memory, you type the command LOADHIGH, then follow it with the program command you would normally enter at the command prompt. For example, to load the DOSKEY program into a UMB, the following command can be used:

```
LOADHIGH DOSKEY
```

Any parameters or switches the command uses may be included, just as they would on the command line. If there is no UMB large enough for the program, LOADHIGH will load the program in low memory. Unfortunately, LOADHIGH does not indicate which memory area is used, so it is necessary to use the MEM /CLASSIFY command to see whether the program went into a UMB or into low memory.

To see the effects of the DEVICEHIGH and LOADHIGH commands, compare Figure 4-4 with Figure 4-5.

```
Conventional Memory :

  Name                Size in Decimal        Size in Hex
-------------      ---------------------   -------------
  MSDOS                65824     ( 64.3K)      10120
  HIMEM                 3200     (  3.1K)        C80
  EMM386                8400     (  8.2K)       20D0
  COMMAND               4704     (  4.6K)       1260
  FREE                    64     (  0.1K)         40
  FREE                571968     (558.6K)      8BA40

Total  FREE :         572032     (558.6K)

Upper Memory :

  Name                Size in Decimal        Size in Hex
-------------      ---------------------   -------------
  SYSTEM              164864     (161.0K)      28400
  ANSI                  4192     (  4.1K)       1060
  SK                   69408     ( 67.8K)      10F20
  DOSKEY                4128     (  4.0K)       1020
  FREE                 20464     ( 20.0K)       4FF0

Total  FREE :          20464     ( 20.0K)

Total bytes available to programs (Conventional+Upper) :     592496    (578.6K)
Largest executable program size :                            571776    (558.4K)
Largest available upper memory block :                        20464    ( 20.0K)

   1310720 bytes total contiguous extended memory
         0 bytes available contiguous extended memory
   1059840 bytes available XMS memory
           64Kb High Memory Area available
```

Figure 4-5 — MEM Command Output with ANSI.SYS and DOSKEY Loaded High.

Both figures were produced on a 386SX PC with two megabytes of RAM. Both have one device driver (ANSI.SYS), and two TSRs (DOSKEY and SIDEKICK 1.5) loaded. In Figure 4-4, HIMEM.SYS and EMM386.EXE are not loaded, and the upper memory area is not used. In Figure 4-5, HIMEM.SYS and EMM386.EXE are loaded and the upper memory area is used.

There are several significant differences in these two figures. To begin with, in Figure 4-4, where HIMEM.SYS and EMM386.EXE are not loaded, there are 483,376 bytes available for running programs in conventional memory (the difference between "bytes available" and "largest executable program size" will be discussed shortly). Some of the low memory is occupied by DOS, ANSI.SYS, COMMAND.COM, and DOSKEY, and SIDEKICK. There are 1,310,720 bytes of extended memory free, and no space available in the upper memory area. In Figure 4-5, where HIMEM.SYS and EMM386.EXE are loaded, there are 572,032 bytes available for running programs in conventional memory. Thus, by using UMBs for device drivers and TSRs, 88,656 bytes of memory have been reclaimed for running programs. Notice, though, that EMM386.EXE uses 8400 bytes of conventional memory. If you are not loading device driver's or TSRs that total at least this much memory, then you may be better off not loading EMM386.EXE and foregoing the use of UMBs.

Figure 4-5 has a section labeled "Upper Memory" that does not appear in Figure 4-4. This section shows that the device driver and TSRs have successfully been loaded into UMBs, and that there is still 20,464 bytes of memory available for additional device drivers and/or TSRs. If there were no other device drivers or TSRs you wanted to use, you could use this space to increase the buffer size of the DOSKEY program. There are also 164,864 bytes of the upper memory area that are labeled SYSTEM. This represents the amount of upper memory being used by the traditional services assigned to the reserved memory area — the ROM-BIOS, the video card, and other system functions. These system functions are always stored in the upper memory area, but do not appear in Figure 4-4, because the MEM /CLASSIFY command only shows the upper memory area when HIMEM.SYS and EMM386.EXE are loaded.

In Figure 4-4, 483,376 bytes are designated as available, but only 483,120 bytes are designated as the largest executable program size. This is due to the fact, mentioned earlier, that programs require contiguous memory in which to load. The difference of 256 bytes represents memory that is not contiguous. There can be one or more gaps between the items loaded into low memory and these gaps total 256 bytes. The gaps contain unused memory, and are therefore available, but they are so small as to be virtually unusable.

In Figure 4-5, the total number of bytes free in the upper memory area is 20,464. The total conventional plus upper memory available is 592,496 bytes, and the largest executable program size 571,776 bytes. The largest available UMB is 20,464 bytes. This is a useful piece of information — the total number of bytes available in UMBs are all in one UMB. This means that any item up to 20,464 bytes will fit into the UMB. If the 20,464 UMB bytes were spread out in several UMBs, then the largest item that could go into a UMB would be less than the total number of UMB bytes available.

If the total bytes in UMBs (20,464) is subtracted from the bytes available to programs (592,496), the result is 572,032; this is the total amount of free conventional memory. Subtracting the largest executable program size (571,776) gives 256 bytes — the same number of bytes lost to gaps between the items in low memory as we saw in Figure 4-4.

The final difference between the two memory configurations is in the available extended memory. Both figures show that there are 1,310,720 bytes total contiguous extended memory. However, Figure 4-4 shows that all of this is available, while Figure 4-5 indicates that only 1,059,840 bytes are available. The difference, 250,880 bytes, is the amount of extended memory that has been used by EMM386.EXE to back-fill the UMBs. These 250,880 bytes are now referenced with the upper memory area addresses of the UMBs, and are therefore treated as conventional memory. They are no longer available for use as extended memory.

The "64Kb High Memory Area available" shown in Figure 4-5 is discussed in a later section, Managing Extended Memory.

In summary, to maximize the use of conventional memory on a 386 PC that has at least one megabyte of RAM, perform the following steps:

1. Place the `DEVICE=C:\DOS\HIMEM.SYS` command in the CONFIG.SYS file.
2. Place the `DEVICE=C:\DOS\EMM386.EXE /NOEMS` command in the CONFIG.SYS file.
3. Place the `DOS=UMB` command in the CONFIG.SYS file.
4. Use DEVICEHIGH rather than DEVICE in CONFIG.SYS to load device drivers. Be sure the DEVICEHIGH commands come after the commands in steps (1) through (3).
5. Use the LOADHIGH command to place any TSRs in UMBs.
6. If one or more of the device drivers does not work, or the computer locks up, then use the DEVICE command to load the problem driver(s) into low memory, and the MEM command to determine their size requirements. Then try using the DEVICEHIGH command again, this time with the SIZE parameter.

If you have applications that need expanded memory, you should use the /RAM switch instead of /NOEMS on the EMM386.EXE line. This sets up the page frame required for expanded memory.

When you have more device drivers and/or TSR programs than will fit into the available UMBs, there are usually several possible configurations. For example, suppose that you have three TSRs you want to put into high memory, one which is 20K, one 30K, and one 40K. Now suppose that there is one 80K UMB available. You cannot fit all three into the UMB, since they total 90K, but any two of them will fit. Since the one that's left will have to go into conventional memory, it is to your advantage to make that one the smallest. Therefore, you should put the 30K and 40K TSRs into high memory, and the 20K TSR into low memory. This is a simple example, but it illustrates the point that it is worth examining carefully what is going into high memory and what is going into low memory, when you cannot put all of your device drivers and TSRs into high memory.

If you have a 286 PC with one megabyte or more of memory, it is still possible to make use of the UMBs, but to do so requires the purchase of a memory manager that will work on a 286 PC, since EMM386.EXE works only on 386 and later PCs.

Another technique that can be used to provide a small amount of additional conventional memory is the use of the INSTALL command. The INSTALL command goes in the CONFIG.SYS file, and is used to load TSRs. Normally, when a TSR is loaded, a copy of the DOS environment is loaded with it. Many TSRs require this, but some do not. In particular, the DOS programs FASTOPEN, NLSFUNC, KEYB, and SHARE do not need a copy of the environment. INSTALL loads TSRs without storing a copy of the environment, thereby slightly reducing the amount of memory they use. To use INSTALL with a FASTOPEN command that stores up to 100 entries for drive C:, put the following command in the CONFIG.SYS file:

```
INSTALL FASTOPEN C:=100
```

This replaces the FASTOPEN command that would appear in the AUTOEXEC.BAT file. Most third party TSRs will not work with INSTALL, and trying to use INSTALL with them may lock up the computer. Consequently, if you are going to experiment with this, make sure you have bootable floppy disk, so you can boot your PC if the CONFIG.SYS file on your hard drive is locking up the PC.

Managing Expanded Memory

If you plan on using expanded memory, there are two things to remember:

- on 8088 and 80286 PCs, expanded memory requires a special add-on board, and software that comes with the board
- Expanded memory can only be used with software that has been specifically designed to use it. Other software will ignore expanded memory.

PCs based on the 80386 and later chips have the hardware that is required to use expanded memory built in. Consequently, all they need is the proper software to convert extended memory to expanded memory. Therefore, it is not necessary to buy add-on boards to put expanded memory on an 80386 PC. Furthermore, since the hardware portion of expanded memory management is built in to the CPU, and all 80386 PCs use the same CPU chip, regardless of the PC manufacturer, the same expanded memory management software will work on all 80386 PCs.

For this reason, DOS 5.0 includes an expanded memory manager for 80386 PCs: EMM386.EXE.

Note: In order to use EMM386.EXE, the HIMEM.SYS driver must be installed ahead of it, in the CONFIG.SYS file.

EMM386 is what is known as an "expanded memory emulator" because it uses extended memory to emulate expanded memory. In other words, EMM386 can be used to trick the computer into thinking that it's extended memory is actually expanded memory. If you are using another expanded memory manager, you should not use EMM386. You should decide which suits your needs better, and eliminate the other one.

In order to use expanded memory, a 64K page frame must be set up to map the expanded memory into conventional memory, and an expanded memory manager must then handle the mapping. These two tasks are provided by EMM386. There are numerous switches that can be used with EMM386 to specify the precise location in memory of where the page frame is located, and how EMM386 behaves. These switches are rarely needed, and their use is quite technical. They are therefore discussed in the Advanced Topics section of this chapter. In general, allowing EMM386 to use the default settings for these switches works best. However, there are some other switches that are more commonly used, and they will be discussed here.

EMM386 must be loaded when the computer boots, it cannot be loaded later. To do this, it is included in a DEVICE= command in the CONFIG.SYS file. After it is loaded, however, two aspects of its functioning may be controlled from the command line. The command EMM386 is used to do this, with one of the following switches:

```
ON

OFF

AUTO

W=ON

W=OFF
```

The ON switch turns the EMM386 driver on, if it has been deactivated using the OFF switch. When the driver is first loaded, it is ON by default.

The OFF switch deactivates the EMM386 driver. This does not restore expanded memory to the pool of extended memory, but it does make the PC appear to programs as though it does not have any expanded memory. This is useful if the presence of an expanded memory manager conflicts in someway with a program.

The AUTO switch causes the EMM386 driver to be active whenever a program calls for expanded memory, but inactive at all other times. This is useful if you have some programs that require expanded memory, and others that conflict with it, and want to run both types of program during the same session.

There is a special math co-processor from the Weitek company that is popular with some PC users. For EMM386 to provide expanded memory support for this co-processor, the W=ON switch must be specified. To turn off support for the co-processor, use the W=OFF switch. This switch is OFF by default.

All of the above switches are entered with the EMM386 command. They are only effective if the EMM386 driver was loaded. The rest of the switches discussed must appear on the DEVICE=EMM386.EXE line in the CONFIG.SYS file.

The amount of extended memory to be converted to expanded memory can be specified in the CONFIG.SYS file, by including the desired number of kilobytes as a parameter on the DEVICE command line. For example, to set up 512K of expanded memory, the following command should be included in the CONFIG.SYS file:

```
DEVICE=EMM386.EXE 512
```

The amount of memory specified may be anywhere from 16K to 32,768K. Memory should be specified in multiples of 16. If the number used is not a multiple of 16, then EMM386 will automatically round the number down to the nearest multiple of 16. For example, if you specified 520, EMM386 would round it down to 512. If no value is specified, EMM386 uses 256. It is a good idea to make sure you use a multiple of 16 when specifying the amount of expanded memory, because it is then obvious how much expanded memory is actually available.

Under some circumstances, you may want to convert as much of your extended memory to expanded as possible, but you must be sure that some minimum amount of extended memory remains available. This can be done with the L=xxx switch, where **xxx** represents amount of extended memory that will be available after you load EMM386. The default value for this switch is `0`. However, this does not mean that if the switch is not specified, all extended memory is converted to expanded. Rather, it means that whatever happens, the computer will be left with the specified amount of extended memory — assuming that it has that much to begin with.

To make the best use of your memory, you should only allocate as much expanded memory as a program requires, and leave the rest as extended. If you are running Microsoft Windows version 3.0 or later in 386 enhanced mode, then Windows can emulate expanded memory for those programs that need it. Use the Windows emulator rather than EMM386. However, if you have programs that will run outside of Windows that require expanded memory, then you must use EMM386. If you want to use EMM386 to provide access to UMBs, but don't need it as an expanded memory manager, then use the NOEMS switch on the DEVICE=EMM386.EXE line in the CONFIG.SYS file.

Managing Extended Memory

If you want to make use of extended memory, your software must be designed to use it. Software that is not designed to use extended memory will ignore it. Extended memory can only exist on PCs that use 286, 386, and more advanced chips. There are two techniques that software developers can use to design software that takes advantage of extended memory. The first is to build a "DOS extender" into the software, then nothing needs to be done to the system to make the extended memory available.

The second approach is to design the software to conform to the XMS Version 2.0 protocol. If the computer has an extended memory manager installed that also conforms to this protocol, the extended memory will be available to the program. DOS 5.0 provides HIMEM.SYS for this. To install HIMEM.SYS, include the following line in your CONFIG.SYS file:

```
DEVICE=HIMEM.SYS
```

There are several switches that can be used with HIMEM.SYS. These switches are highly technical, however, and are usually best left to their default values. They are discussed in the Advanced Topics section of this chapter.

In addition to providing extended memory management, HIMEM.SYS also makes a special 64K area of extended memory available to DOS, as though it were conventional memory. This area is known as the HMA, or High Memory Area. This terminology is a little confusing, since the region from 640K to 1 megabyte is referred to as the Upper Memory Area, or High Memory. The HMA is not part of the Upper Memory Area. Rather, it is the first 64K of extended memory — the memory that extends from address 1,048,576 (1 megabyte) to address 1,114,112 (64K past 1 megabyte). When HIMEM.SYS is loaded, it is possible to put a substantial portion of DOS itself into the HMA, thereby freeing up additional conventional memory. To load part of DOS into the HMA, include the following line in your CONFIG.SYS file:

```
DOS=HIGH
```

STOP

Note that DOS=HIGH must be listed after the DEVICE=HIMEM.SYS command in CONFIG.SYS.

This command may be used on any PC that runs on a 286 or later chip, and that has extended memory available. This would include all PCs having one or more megabytes of RAM. If you want to load DOS into the HMA, and also provide UMBs, then the DOS command should be:

```
DOS=HIGH,UMB
```

To see the effect of loading DOS high, compare Figure 4-6 with Figure 4-5. Figure 4-6 was generated with the MEM /CLASSIFY command after the DOS=UMB line in the CONFIG.SYS file was modified to include the HIGH switch.

```
Conventional Memory :

  Name                Size in Decimal       Size in Hex
-------------      ---------------------  -------------
  MSDOS               15632     ( 15.3K)      3D10
  HIMEM                1184     (  1.2K)       4A0
  EMM386               8400     (  8.2K)      20D0
  COMMAND              2624     (  2.6K)       A40
  FREE                   64     (  0.1K)        40
  FREE               626240     (611.6K)     98E40

Total  FREE :        626304     (611.6K)

Upper Memory :

  Name                Size in Decimal       Size in Hex
-------------      ---------------------  -------------
  SYSTEM             164864     (161.0K)     28400
  ANSI                 4192     (  4.1K)      1060
  SK                  69408     ( 67.8K)     10F20
  DOSKEY               4128     (  4.0K)      1020
  FREE                20464     ( 20.0K)      4FF0

Total  FREE :         20464     ( 20.0K)

Total bytes available to programs (Conventional+Upper) :    646768   (631.6K)
Largest executable program size :                           626048   (611.4K)
Largest available upper memory block :                       20464   ( 20.0K)

   1310720 bytes total contiguous extended memory
         0 bytes available contiguous extended memory
   1059840 bytes available XMS memory
           MS-DOS resident in High Memory Area
```

Figure 4-6 — MEM Command Output with ANSI.SYS, DOSKEY, and DOS Loaded High.

In Figure 4-6, the line "MS-DOS resident in High Memory Area" has replaced the line "64Kb High Memory Area available" in Figure 4-5, indicating that part of DOS is now using the HMA. Now compare the "Total bytes available to programs (Conventional+Upper)" and "Largest executable program size" lines in the two figures. Figure 4-6 shows that using the HMA for DOS frees up an additional 54,272 bytes for running programs.

In some circumstances, a commercial memory manager such as QEMM-386 may provide even more efficient use of the UMBs and HMA than EMM386.EXE does. In particular, EMM386.EXE uses the HMA to load part of DOS, but this leaves part of the 64K HMA unused and unavailable. QEMM-386, on the other hand, allows you to fully utilize the HMA. Consequently, when running QEMM-386 it may be better to load DOS into low memory and use QEMM-386 to manage the HMA.

Memory Management Summary

While all memory on Personal Computers is physically the same, there are three ways in which that memory can be configured, depending on the type of hardware (CPU and memory boards) installed: conventional memory, which all PCs have; expanded memory, which can be added to any type of PC; and extended memory, which can only exist on PCs that use an 80286 or later CPU.

The advantage of providing as much conventional memory as possible is that all programs can make use of conventional memory, and in general it is the fastest memory for a program to use, since no additional overhead is involved in managing it. DOS 5.0 provides the HIMEM.SYS and EMM386.EXE device drivers, and the DOS command, to provide the largest possible amount of conventional memory for programs.

The advantage of expanded memory is that it can be added to any type of PC, and that there is a fairly large number of software programs that can take advantage of it. The disadvantage is that it is slower to access than either conventional memory and extended memory, because of the complex paging mechanism by which it is implemented. Furthermore, computers and software are evolving away from the use of expanded memory, toward the use of extended memory, so that expanded memory must be seen as a short-term investment. To use expanded memory on an 80286 PC, expanded memory management software must be installed. This software is specific to the particular brand of expanded memory board you purchased. To use expanded memory on an 80386 PC, the EMM386.EXE device driver can be used to convert extended memory to expanded.

The advantages of having extended memory are that it is very fast (for programs that can make use of it) and that it is a good long-term investment, since the industry is evolving towards extended memory as the standard way of using memory over 1 megabyte. The disadvantages are that it cannot be added to 8088 PCs, and that most programs cannot make use of it.

Extended memory can also be used to increase the amount of conventional memory available to programs, by loading DOS into the HMA. On 386 PCs, extended memory may also be used to back-fill the UMBs, which can then be used to load device drivers and TSRs. This makes even more conventional memory available to programs. Extended memory is made available by including the line DEVICE=HIMEM.SYS in the CONFIG.SYS file. Extended memory may be used by Microsoft Windows Version 3.0 or later to improve performance. Loading DOS into the HMA and backfilling UMBs are both accomplished on 386 PCs by also including the lines DEVICE=EMM386.EXE /NOEMS and DOS=HIGH,UMB in the CONFIG.SYS file.

Advanced Topics

There are several topics related to computer memory that are quite technical in nature, and that do not need to be understood for most Personal Computer users. However, for highly technical users this knowledge can help you to fine-tune the configuration of a PC. You must understand the information presented earlier in this chapter in order to follow this material.

Understanding Memory Addresses

So far, addresses have simply been referred to as numbers that tell DOS where a piece of information lies. However, the internal workings are not quite that simple because the original PCs were only capable of manipulating 16 bits at a time. Consequently, they could only directly manage a 16-bit address. However, that would have limited the PCs to 2^{16}, or 65,536 bytes (64K) of memory. In order to achieve a 20 bit address, a scheme known as "segmented addressing" was used. As its name implies, segmented addressing breaks the entire memory area down into segments. A specific address then consists of two numbers: the 4-bit segment address, also known as the base address, and a 16-bit offset address, which indicates how far into the specified segment a particular address lies.

To envision this, think of having 10,000 boxes, each of which must be assigned a unique numeric code. But you are limited to using codes that are no more than two digits long. That means each box can only have an "address" that falls between 0 and 99. So to get around the problem, you arrange the boxes into groups. Since you are capable of managing two-digit codes, you can have up to 100 groups, labeled 0 through 99. These "group addresses" are the segment, or base, addresses. After arranging the boxes into these groups, each box within a group is assigned its own two-digit code. Again, up to 100 boxes can be in each group, with "offset addresses" ranging from 0 to 99. Now, to specify a particular box, you provide two numbers: the base address, and the offset address.

When actually referencing a particular box, you must use the offset and base addresses, but for thinking about what you want to put into and remove from the boxes, and how much they can hold, you can imagine that they have one long sequence of 10,000 addresses, ranging from 0 through 9,999. The first box in group 0 has an offset address of 0, but it can be thought of as having an "absolute" address of 1. The first box in group 1 can be thought of as having an "absolute" address of 101, even though the two-digit numbering scheme cannot directly represent the number 101. This is what has been done up to now in this chapter — memory locations have been referred to with their "absolute" address, rather than their segment plus an offset address.

The segment part of the address in an IBM PC uses 4 bits, so there are 16 segments (2^4) in the memory space of a PC. Since the segment address uses up only four bits, and the total address is 20 bits, the offset address uses 16 bits. This means that each segment is 65,536 bytes long. The result of 16 segments, each of which contains 65,536 bytes, is the total address space of a PC: 16 times 65,536 equals 1,048,576, which is one megabyte.

Switch Settings for the EMM386.EXE Device Driver

The commonly used switches for EMM386 are discussed earlier in this chapter, and all of the switches are summarized in Chapter 10. The switches discussed in the following paragraphs provide a fine level of control over precisely how EMM386 sets up the page frame, as well as other aspects of its behavior. None of these switches are required to be specified — they all have default values that

STOP

are used if they are not specified. In most cases the default values are acceptable. Except where noted otherwise, any of the EMM386 switches may be used with any of the other switches. Caution should be exercised when using these switches. Valid settings not only depend on the values specified for the switch, but in some cases on the particular hardware configuration. Setting them improperly may cause the PC to lock up. You should have a very good understanding of how expanded memory works before adjusting these switches.

When EMM386 is used to manage expanded memory, it creates a 64K page frame in the upper memory area. It is no coincidence that 64K is also the size of a memory segment. By making the page frame 64K, a single address can be stored as the segment address of the page frame, and any other address reference to the page frame is seen as the offset.

The page frame is the area that expanded memory pages are mapped into as they are needed. Each page is 16K, so there are four of them in the page frame. If left to itself, EMM386 will determine an appropriate location within the upper memory area for the page frame to be located. However, there are three switches that can be used to tell EMM386 precisely where to put the page frame. Since EMM386 can only create one page frame, these switches are mutually exclusive — you may use only one of them. Addresses for the page frame must start on a 16K (16,384 byte) boundary — they must be a multiple of 16,384. In the discussion that follows, you will see that all valid values for the page frame address meet this condition.

The first of these switches is the Mx switch, where **x** represents a code that corresponds to an address. The codes, their corresponding hexadecimal addresses, and full decimal addresses, are shown in Table 4-1. Note that the hex addresses are lacking a final zero: since the address must lie on a 16K boundary, the last digit will always be zero, so the final zero is not specified.

Table 4-1 — Valid EMM386 "M" Switch Codes and the Corresponding Addresses.

Code	Address	(Hex Dec)
1	C000	786,432
2	C400	802,816
3	C800	819,200
4	CC00	835,584
5	D000	851,968
6	D400	868,352
7	D800	884,736
8	DC00	901,120
9	E000	917,504
10	8000	524,288
11	8400	540,672
12	8800	557,056
13	8C00	573,440
14	9000	589,824

To use the M switch to specify that the page frame begin at address C000 (hex), the following command would be used:

```
DEVICE=EMM386.EXE M1
```

Notice that the last five addresses in Table 4-1 are in the region between 512K (524,288) and 640K (655,360). These settings should only be used on PCs that have 512K of memory. If they were used on a PC that had 640K, the page frame would be placed in usable memory. This should be left available for programs.

The second switch for setting the page frame uses the address directly. This has the same effect as the M switch, but is more readable — you don't have to refer to a table to determine what the actual address is. The syntax for this switch is:

```
DEVICE=EMM386.EXE FRAME=address
```

The address must be given in hexadecimal. Valid values are any of the hex values shown in Table 4-1.

The third switch for setting the page frame is the /Pxxxx switch, where **xxxx** represents the address, specified in hex. Again, valid values are those hex addresses shown in Table 4-1. For example, to use this switch to specify that the page frame begin at address DC00 (hex), the following command would be used:

```
DEVICE=EMM386.EXE /PDC00
```

The LIM-EMS Version 4.0 specification allows for more than just four pages to be used for swapping expanded memory into and out of conventional memory. EMM386 has a switch that can be used to specify the segment address of a specific page. The syntax for this switch is:

```
DEVICE=EMM386.EXE Px=address
```

Where **x** represents the page number. This may be any number from 0 through 255. Again, valid values for the address are those hex values in Table 4-1. In order to maintain compatibility with the earlier version (3.2) of LIM-EMS, the addresses for pages 0 – 3 must be contiguous. It is best, therefore, to allow EMM386 to set these page addresses.

The Px=address switch may not be used if any of the switches for setting the page frame address itself are used (the Mx, FRAME=, and /Pxxxx switches).

Normally, EMM386 senses what addresses are already in use and avoids them when setting up the page frame. However, there may be circumstances under which you want to specifically prevent EMM386 from using a particular range of addresses. This is done with the X switch, which has the following syntax:

```
DEVICE=EMM386.EXE X=address1-address2
```

The addresses are specified in hexadecimal, and may be in the range from A000 (655,360, or 64K) through FFFF (1,047,552, or 1023K). Again, the trailing zero is not specified. The addresses should be multiples of 4k (4096). If they are not, they are rounded down to the nearest multiple of 4k. If this switch is used in conjunction with the I switch, which is discussed next, and the ranges specified in the two switches overlap, then the X switch takes precedence.

The opposite of the X switch is the I switch, which specifies a range of addresses that should be included in an EMS page, or for RAM. The syntax of this switch is:

```
DEVICE=EMM386.EXE I=address1-address2
```

The address requirements are the same as for the X switch.

Another enhancement from LIM-EMS version 3.2 to 4.0 is the ability to backfill memory below the 640K boundary, and to map memory below 640K for page swapping. The B switch on EMM386 is used to specify the lowest segment address that should be used for swapping pages. Valid values are from 1000 hex (65,536 decimal, or 64K) to 4000 hex (262,144 decimal, or 256K). The trailing zero is left off in the hex number. The default value is 4000 hex, or 256K. The syntax is:

```
DEVICE=EMM386.EXE B=address
```

If left to itself, EMM386 sets up 256K of expanded memory and leaves the rest of extended memory alone. As discussed earlier, you may specify more than 256K of expanded memory by including the number of kilobytes you want on the DEVICE line. However, you may be more concerned with how much extended memory is left than with how much expanded memory you have. In this case, the L=kilobytes switch specifies the minimum amount of extended memory that must be left after EMM386 is loaded. The default value is `0`.

Specifying a particular number here does not cause all of the extended memory beyond that number to be converted to expanded memory. Rather, it says that no matter what else happens, at least this much extended memory will be left. The syntax is:

```
DEVICE=EMM386.EXE L=kilobytes.
```

While LIM-EMS 4.0 is not itself a multi-tasking operating system (an operating system in which multiple programs may be run simultaneously, as they can under DESQview and Windows), it does provide a large number of services that programmers can use to develop multi-tasking environments. These services allow multiple applications and tasks to run simultaneously in expanded memory. When expanded memory is used in a multi-tasking environment, EMM386 sets up alternate register sets. Registers are special memory locations within the CPU that are used for storing intermediate values as they are operated on, and as they move to and from memory. Setting up alternate register sets considerably speeds up task switching in a multi-tasking environment. By default, EMM386 sets up seven fast alternate register sets. This number may be adjusted with the A switch:

```
DEVICE=EMM386 A=numsets
```

The value of **numsets** may be anywhere from 0 to 254. Each alternate register set requires about 200 bytes of memory.

When an application requests expanded memory, that application is assigned a "handle" by EMM386. A handle is simply a number that identifies the application. All communication is then done by referencing the handle number. The number of handles that may be used is defined with the H switch:

```
DEVICE=EMM386.EXE H=numhandles
```

The value of numhandles may be anywhere from 2 through 255; the default value is 64.

An integral part of a PC is something known as the DMA (or **D**irect **M**emory **A**ccess) chip. DMA refers to the ability for disk drives to move information directly to and from memory, without requiring the intervention of the CPU. This greatly speeds up the operation of the computer. In order to make this work efficiently, DMA transfers can be buffered, so that you do not have to wait for the disk drive to be ready before the data transfer can begin. EMM386 sets up 16 kilobytes of memory for this purpose, but this number can be adjusted with the D switch:

```
DEVICE=EMM386.EXE D=kilobytes
```

The number of kilobytes may be anywhere from 16 to 256. In general, the number used should reflect the size of the largest DMA transfer that will occur between memory and a hard drive.

Switch Settings for the HIMEM.SYS Device Driver

As discussed earlier in this chapter, the HIMEM.SYS driver is used to make the HMA (High Memory Area) available to programs (provided the programs are designed to make use of it). There are only a few programs that can make use of the HMA, but among those is the DOS=HIGH command in DOS 5.0. Only one program at a time can use the HMA, however, so the program that will use it most efficiently should be the one that has access to it. For example, if two programs could use the HMA, and one uses 60K, while the other only uses 20K, then it is preferable for the first one to use it. This can be controlled with the following switch:

```
DEVICE=HIMEM.SYS /HMAMIN=xxx
```

The **xxx** represents kilobytes, and may be anywhere from 0 to 63. The HMAMIN switch specifies that a program must use at least xxx kilobytes of memory before it is granted access to the HMA. Thus, if xxx is 60, and a program that uses only 50 kilobytes attempts to use the HMA, it will not be allowed to. When a larger program subsequently requests access to the HMA, it will still be available. If this switch is not specified, the default value is 0.

In order to manage memory according to the XMS 2.0 specification, extended-memory-block handles are assigned. The number of handles that HIMEM.SYS may use simultaneously is 32, but this number can be adjusted with the /NUMHANDLES switch:

```
DEVICE=HIMEM.SYS /NUMHANDLES=x
```

The value of **x** may be anywhere from 1 through 128. Each handle uses six bytes of memory, so specifying 128 handles would increase the memory used by HIMEM.SYS by 576 bytes over the default (128 – 32 = 96; 96 X 6 = 576).

In rare cases, it is necessary to adjust the amount of extended memory allocated to the Interrupt 15 (hex) interface. Discussion of this is beyond the scope of this book, but the amount of memory allocated for this purpose may be adjusted with the /INT15=xxx switch. The following command reserves 64K of memory for the Interrupt 15h interface:

```
DEVICE=HIMEM.SYS /INT15=64
```

The number of kilobytes specified may be anywhere from 64 through 65,535. If a value less than 64 is specified, the value is adjusted down to 0. The default value is 0.

Your computer contains a device known as the "A20 handler." This device provides the computer with access to the HMA. There are a variety of ROM-BIOS chip sets that can be purchased, and each one has its own A20 handler. Different PC manufacturers use different chip sets. HIMEM.SYS must turn on the A20 handler in order to use it. In most cases, HIMEM.SYS is able to correctly detect the A20 handler in use, but in some cases it may be necessary to manually specify which one your machine has. This is done with the /MACHINE switch. The syntax for this switch is:

```
DEVICE=HIMEM.SYS /MACHINE:xxxx
```

The value of **xxxx** may be either a code or a number. The codes and numbers, and the corresponding computer or ROM-BIOS manufacturer, are shown in Table 4-2.

Table 4-2 — A20 Handler Codes and Numbers

Code	Number	A20Handler
at	1	IBM PC/AT
ps2	2	IBM PS/2
ptlcascade	3	Phoenix Cascade BIOS
hpvectra	4	HP Vectra (A and A+)
att6300plus	5	AT&T 6300 Plus
acer1100	6	Acer 1100
toshiba	7	Toshiba 1600 and 1200XE
wyse	8	Wyse 12.5 Mhz 286
tulip	9	Tulip SX
zenith	10	Zenith ZBIOS
at1	11	IBM PC/AT
at2	12	IBM PC/AT (alternative delay)
css	12	CSS Labs
at3	13	IBM PC/AT (alternative delay)
philips	13	Philips
fasthp	14	HP Vectra

If the machine you are using is not specified in Table 4-2, and does not use a ROM-BIOS that is listed there, it may be included in a file named README.TXT that is supplied with DOS 5.0. This file is continuously updated to add new PCs to the list. The default value for the MACHINE switch, if HIMEM.SYS does not recognize your machine, is "at," which is the same as 1.

It is possible that the A20 handler will have already been turned on when the HIMEM.SYS driver is loaded. In this case, HIMEM.SYS will take over control of the A20 handler. However, under some circumstances this is not desirable. The following switch may be used to control how HIMEM.SYS behaves in relation to the A20 handler:

```
DEVICE=HIMEM.SYS /A20CONTROL:xx
```

Where **xx** is either ON or OFF. The default value is on, in which case HIMEM.SYS takes over control of the A20 handler even if it was already turned on. If off is specified, HIMEM.SYS takes over control of the A20 handler only if it was off when HIMEM.SYS was loaded.

On some computers that have extended memory, some of that memory may be used as "shadow RAM." Shadow RAM refers to RAM that has had the contents of some of the ROM placed into it. The reason for placing the contents of ROM into RAM is that RAM is much faster memory — it can transfer its contents to and from the CPU faster than ROM can. Consequently, loading the contents of ROM into RAM speeds up the performance of the computer, at the expense of losing some extended memory.

If your computer sets up shadow RAM, it may be possible to turn off the shadow RAM, using the SHADOWRAM switch. This results in the ROM being placed back in service, which will slow down the computer's performance, but it also returns the memory used as shadow RAM to the extended memory pool. The syntax for this switch is:

```
DEVICE=HIMEM.SYS /SHADOWRAM:ON
DEVICE=HIMEM.SYS /SHADOWRAM:OFF
```

Specifying OFF disables the shadow RAM; specifying ON leaves the shadow RAM in place. If your computer does not automatically set up part of extended memory as shadow RAM, this switch cannot be used to set it up. This switch is used only to remove shadow RAM from computers that have this feature built in. On computers that have less than two megabytes of memory, the default value is OFF; on computers with two megabytes of memory or more, the default value is ON.

The last switch that may be used with HIMEM.SYS is the /CPUCLOCK switch. This switch tells HIMEM.SYS whether it should affect the clock speed of your computer. On most computers, HIMEM.SYS does not affect the clock speed, but on some, you may notice that the clock speed changes when HIMEM.SYS is loaded. In this case, specifying ON may solve the problem. However, enabling this switch will slow down the performance of HIMEM.SYS. The syntax for this switch is:

```
DEVICE=HIMEM.SYS /CPUCLOCK:ON

DEVICE=HIMEM.SYS /CPUCLOCK:OFF
```

More on RAM

For most purposes it is sufficient to consider all RAM as being created equal. However, there are some differences among various RAM chips. The most commonly discussed difference is the access time — how long it takes to read or write information to and from a chip. Access times are typically measured in nanoseconds. A nanosecond is 10^{-9} seconds, or one billionth of a second. When the first PCs appeared, access time wasn't much of an issue. Access times of memory chips was around 150 nanoseconds, which was plenty fast for the CPUs in use at that time. However, these memory chips were unable to keep up with the faster CPUs that developed, so faster memory chips were designed. This evolutionary process will continue as CPUs get faster and faster.

In addition to different access times, there are two physically different types of RAM chips that may be installed in a computer: SRAM and DRAM. These terms stand for **S**tatic RAM and **D**ynamic RAM. DRAM is the most common form; it utilizes the charge stored on a capacitor to maintain its contents. Capacitors inevitably have some leakage, so DRAM must be continually refreshed. That is why Dynamic RAM is so named — its contents are dynamically changing. DRAM is less expensive to manufacture, but also has a longer access time, than SRAM. RAM does not need to be refreshed, and has a faster access time, but is more expensive to produce.

Real, Protected, and Enhanced Chip Modes

When the first PCs were made, and there was only one kind of CPU that a PC could have, the term "mode" was not used. However, when the 80286 chip was developed, it had capabilities that went far beyond the original 8088/8086 chips. These capabilities included a larger address bus (24-bit, rather than 20-bits) which allowed it to recognize up to 16 megabytes of memory. However, all of the software up to that time had been designed to use the 20-bit address found on the 8088/8086 chips. To permit the 80286 chip to run the existing software, it was designed to operate in two "modes." The first mode, called "real" mode, emulates the 8088/8086 addressing scheme. This permits the 80286 to run 8088/8086 software, but also gives it the same 1 megabyte limitation of addressable memory. Real mode is so named because the address generated by software is translated directly into the real address of a memory location.

In order to make it possible for future software to utilize the larger memory capacity of the 80286, a second mode was implemented, termed "protected mode." In protected mode, the chip uses the full 24-bit address.

The 80286 can be shifted back and forth between real and protected mode under software control. Consequently, as software developed that made use of protected mode, all it had to do was send a signal to the chip to tell it which mode was desired.

In protected mode, software does not generate a real address. The offset part of the address is real — it refers to a specific offset within the specified segment — but the segment address is actually a reference to a lookup table, where the real segment address is stored. This scheme is not simple, but it provides an extremely flexible means for efficiently managing memory under very complex software requirements. In protected mode, different areas of memory may be set up for different programs, all of which are running at the same time. These areas of memory must be protected from each other — one program must not be allowed to intrude on the memory assigned to another program. Protected mode is also referred to as "standard" mode. Further discussion is beyond the scope of this book, but it is interesting to see where the terms real and protected come from.

The 80386 and 80486 chips also support real and protected mode operations. As with the 80286, real mode permits them to run all of the software developed for the IBM PC. However, in protected mode, these chips use their full 32-bit address. This permits them to address up to 4 gigabytes of memory. These chips also support a third mode, known as "386 enhanced" mode. 386 enhanced mode uses essentially the same addressing scheme as protected mode. However, at the point at which protected mode has retrieved its segment address from the lookup table and is ready to produce a physical memory address, 386 enhanced mode references into yet another lookup table.

This lookup table may contain addresses for as much as four times the RAM that actually exists in the computer. The table keeps track of which addresses correspond to actual RAM, and which do not. Those that do not are assigned to a swap file on the computer's hard disk. This swap file is used as virtual memory, so when a piece of data is read from or written to one of these addresses, it actually goes to and from the disk.

Swap files are vastly slower than electronic memory, but they do provide the ability to perform operations on computers that do not have the necessary RAM. For example, if a computer has four megabytes of RAM, then programs requiring up to 16 megabytes of memory could be run in 386 enhanced mode. Of course, this assumes there is enough free disk space to create a 12 megabyte swap file.

Windows version 3.0 and later uses all of these techniques to maximize the performance of each computer it runs on.

Chapter 5

Understanding and Managing Files and Directories

All of the information a computer uses, whether it is programs that the computer runs or data that the computer works on, is stored in disk files. The files on a disk may be organized into groups, called directories. Directories are groupings that you can create and modify at will, allowing you to organize your files into a structure that reflects the way you work, and the ways in which you are likely to use your files. This chapter explores the file and directory features of DOS 5.0, and explains the DOS commands used for managing them.

Files

A file is a physical structure stored on a magnetic disk. It contains information that the computer can read and manipulate. In order to keep track of files, each one is assigned a name. When you purchase a new software program, that program will have one or more files whose names were assigned by the software manufacturer. When you save information that you have entered into the computer — say a letter in a word processor — you provide a name for the file in which it will be saved. That name is then stored on the disk, along with the information contained in the file. When you want to retrieve that information, you specify the name of the file and the computer knows where to look for it.

Since files are storage places for information that will be loaded into memory, where the computer can work with it, the information in files should be stored in a way that reflects the structure of electronic memory. As discussed in Chapter 4, electronic memory consists of millions of tiny switches, which can be either on or off. These switches are referred to as bits and a bit is the smallest amount of information that a computer can store.

A bit that is on is interpreted by the computer as a one. A bit that is off is interpreted by the computer as a zero. Since a single bit can only mean one of two things, computers work with groups of eight bits in order to represent complex information. This group of eight bits is called a byte. Since electronic memory uses information stored as a series of bytes, a computer file also stores information as a series of bytes. Each byte represents a binary number. Depending on the type of information that is stored, the computer can interpret those numbers literally, as numbers, or as characters. If the file contains characters, then the numbers are translated into their corresponding characters according to a table known as the ASCII (or extended ASCII) table. For example, the letter A is represented by the number 65, which in binary (base two) is 01000001. Consequently, when you retrieve a file that shows the letter A on your screen, there is a corresponding location in the file that stores the binary number 01000001.

File sizes are measured in the numbers of bytes they contain. It takes a large number of bytes to store useful information, so file sizes are frequently referred to in terms of kilobytes and megabytes. A kilobyte is 1024 bytes; a megabyte is 1,048,576 bytes. The term kilobyte literally means 1000 bytes, but the computer does everything in multiples of two (since it is operates on the base two number system). 1024 is two raised to the tenth power (2^{10}) and since this is approximately 1000, 1024 bytes is referred to as a kilobyte. Similar reasoning leads to the term megabyte, which is approximately one million bytes. Thus, when you hear of a file being "100K," it means 100 kilobytes (100 X 1024, or 102,400 bytes).

File Names

A DOS file name consists of two parts: a name part, and an extension part. Together, they are referred to as the file name or filespec. The name may contain up to eight alphanumeric characters, and it is required. The extension may contain up to three alphanumeric characters, and is not required. The legal alphanumeric characters that may appear in a file name are shown in Table 5-1.

Table 5-1 — Legal Characters for Use in File Names.

Symbol	Name
A to Z	Letters of the alphabet (not case sensitive)
0 to 9	Numbers
_	Underscore
^	Caret
$	Dollar sign
~	Tilde
!	Exclamation mark
#	Number sign
%	Percent sign
&	Ampersand
-	Hyphen
{	Left curly bracket
}	Right curly bracket
(	Left parenthesis
)	Right parenthesis

DOS ignores the case of an alphabetic character in a file name, so that A and a are treated as the same letter. Only the symbols shown in Figure 5-1 can be used in file names. File names may not contain commas or slashes. If an extension is specified, the name part and the extension part are separated by a period. This is the only place a period may be used in a file name. There are also a few special names reserved by DOS, which cannot be used for file names. These are: CLOCK$, CON, AUX, COMx (where x is 1, 2, 3, or 4), LPTx (where x is 1, 2, 3, or 4), NUL, and PRN.

With a few exceptions, any file can be assigned any combination of name plus extension, as long as the complete file name is unique within its directory. An example of a complete file name is:

```
ROBERT.LET
```

In this example, the name is ROBERT and the extension is LET.

File name extensions are generally used to indicate the type of information stored in the file. For example, all word processing files that contain letters could be given the extension LET, and all files that contain memos could have the extension MEM. In most cases, there are no rules about this, so you may make up whatever extensions make sense to you. Table 5-2 shows a list of commonly used file extensions.

There are a few types of files that are required to have particular extensions in order for them to be usable by the computer. These fall into two categories: extensions required by DOS, and extensions required by software.

The extensions required by DOS arise from the fact that there are only two types of files that can exist in the computer: program files and data files. A program file contains instructions that the computer executes, while a data file contains information that a program can work with. DOS interprets the contents of each of these file types differently. If a file contains a program, then DOS loads the entire program into memory, then tells the CPU to start executing the instructions in the program. The CPU then reads the first byte in the file, interprets it as an instruction, and carries it out. It then goes back to the file (in memory), reads the next item, interprets that as an instruction, and carries it out. This continues until the last instruction in the program is carried out, at which time the program returns control of the CPU to DOS.

Data files, on the other hand, do not contain program instructions. They contain information that programs can manipulate. For example, the file ROBERT.LET could contain a letter written to someone named Robert, created with a word processor. When the word processor edits that file, it tells the CPU to interpret the contents of the file as data rather than as instructions. By distinguishing between program files and data files, the same set of binary numbers can be used to mean two different things to the CPU. This is necessary, because the computer is limited to only 256 different codes that it can use. (The number 256 is 2^8 — since a byte contains eight bits, 255 is the largest number that can be represented in a single byte. A zero is also a code, and this results in 256 possible codes).

In order to distinguish between program files and data files, DOS reserves three file extensions to identify the three types of program files it knows how to execute. These extensions are BAT, COM, and EXE. If you want to run a program with any one of these extensions, you simply type the name of the file on the command line (the extension may be entered, but it is not required and is generally left off), and press Enter↵. DOS then goes to the disk and searches for a file that has that name, followed by one of these three extensions. If DOS finds one, it executes the program in the file. If DOS does not find one, it displays the message "Bad command or file name."

The reason there are three different extensions for executable files is that each one represents a different technique for creating an executable file. Each type requires DOS to load it and interpret its contents in a different way, so by checking the file extension, DOS automatically knows how to load and interpret the file.

The first extension, BAT, indicates that the file is a BATCH file. A BATCH file consists of a list of ASCII-text commands that are executed in sequence, as though they were typed on the DOS command line. These are discussed in detail in Chapter 8.

The COM extension represents an executable file that contains a relatively simple program. Files with a COM extension are limited in size and complexity. Many of the early PC programs were COM files, but as the capabilities of PCs grew, it became necessary to create programs that were more complex than could be produced in the COM format.

The EXE extension represents an executable file that usually contains a more complex program than what you find in COM files. Most software today is distributed as EXE files. The primary difference is that COM files are limited to a maximum of 64K of program space, while EXE files have no such limitation.

You can have three executable files with the same name, but each of the different executable extensions. For example, you could have files named WP.BAT, WP.COM, and WP.EXE. When DOS locates two or three executable files with the same name, and in the same directory, it chooses to run the COM file over either the BAT or EXE file, and it chooses the EXE over the BAT. If you have such a situation, and want to run, say, the BAT file, you must type in the extension as well. DOS will then execute the correct file.

STOP

While the above extensions are intended for use only with executable files, there is nothing in DOS or in the computer that enforces this convention. You could, for example, name a file that contains a word processing document ROBERT.EXE. DOS would permit this, but it is dangerous, since entering the command ROBERT would cause DOS to attempt to load and execute the file. This could cause the computer to lock, requiring you to reboot. These conventions are universally respected.

DOS imposes no conventions on either the name or the extension for data files. However, many software programs automatically assign an extension, and therefore, they only permit you to assign the name. In some cases, you have the option of changing the extension, while in other cases you are required to accept the assigned extension. For example, the Microsoft Excel spreadsheet automatically assigns an extension of XLS to spreadsheet files, unless you include one when you name the file. If you do include an extension, Excel accepts it. In general, it is a good idea to let the programs assign the extensions when they are designed to do so. Commonly used extensions, their typical uses, and what types of programs or software vendors use them, are shown in Table 5-2. Extensions which are listed as ubiquitous, in the last column, are commonly used in a large number of settings.

Table 5-2 — Commonly Used File Extensions.

Extension	Typical Use	Where Used
BAK	Backup copies of files	Ubiquitous
BAS	BASIC language programs	BASIC language
BAT	Batch files	DOS batch files
BIN	Binary object files	Computer languages
C	C language programs	C language
COM	Simple executable programs	Computer programs
DAT	Data files	Ubiquitous
DBF	Database files	dBASE and clones
DOC	Word processing documents	Word processors
DRV	Device driver files	Device driver vendors
DTA	Data files	Ubiquitous
EXE	Large executable programs	Computer programs
LIB	Libraries of language functions	Computer languages
NDX	Database index files	dBASE and clones
OBJ	Object files	Computer languages
OVR	Overlay files	Large computer programs
PAS	Pascal language programs	Pascal language
PRG	dBASE program files	dBASE and clones
RBF	R:BASE database files	R:BASE
SYS	Device drivers	DOS
TXT	Plain ASCII text files	Ubiquitous
WK1	Spreadsheet files	Some spreadsheet programs
WKS	Spreadsheet files	Some spreadsheet programs
XLS	Excel Spreadsheet files	Microsoft Excel
XLM	Excel macro files	Microsoft Excel

It is best to avoid using these common extensions for files you create with programs that do not provide an extension.

File names are frequently used as parameters for DOS commands. For example, the DIR command is used to list the files in a directory. If a file name is passed as a parameter, then only the directory listing for that file is shown. For example, the following command shows the directory listing of the AUTOEXEC.BAT file:

```
DIR AUTOEXEC.BAT
```

Sometimes it is useful to refer to a group of files, rather than a single file. This can be done using the DOS "wild card" symbols. The asterisk (*) is referred to as the "many" wild card. It means "any set of characters can go here." For example, to indicate all files that have an extension of LET you would enter *.LET. This means "select files that have any name, as long as the extension is LET." Files named R.LET, R10.LET, and ROBERT.LET would all be selected. You could also select all files that begin with the letter R and have any extension by specifying R*.*, or files that begin with R and have an extension of LET by specifying R*.LET. You cannot, however, specify *R.*. Anything that appears between the asterisk wild card and the period is ignored, so *R.* selects all files.

The question mark (?) is referred to as the "single" wild card. It means "any single character can go here." For example, specifying R?.LET would select R1.LET and RA.LET, but not ROB.LET. Unlike the asterisk, the question mark wild card can be both preceded and followed by other letters. Question mark and asterisk wild cards can be mixed in a single file specification.

Figure 5-1 shows several legal file specifiers, and some sample file names that would be selected by them.

File Specifications	Sample Files Selected		
?OB.LET	BOB.LET	DOB.LET	FOB.LET
*.DBF	A.DBF	BOOK.DBF	XYZ.DBF
R?B.*	RAB.	RIB.X	ROB.LET
B??K*.*	BEAK.FY	BOOK.LET	BOOK12.DBF
B*.?ET	BAR.BET	B.GET	BOOK.LET

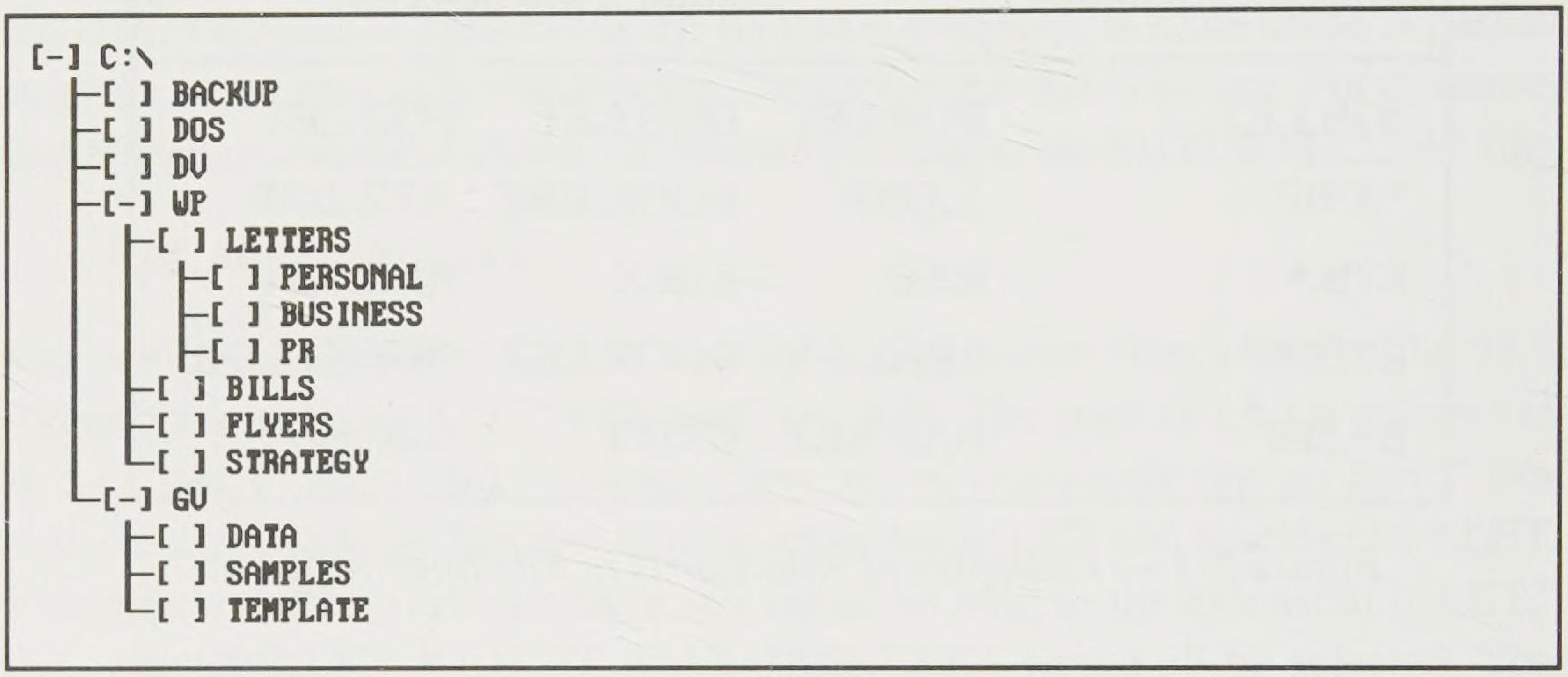
Figure 5-1 — Examples of Wild Cards in File Specifications.

Directories

When the first PCs appeared, most of them only had floppy disk drives. A floppy disk cannot hold very many files, so files were kept organized simply by organizing the floppy disks they were stored on. For example, one floppy disk might contain all of the correspondence you have had with a particular company. However, today most computers have hard disks capable of storing a very large number of files. Trying to keep track of a large number of files by their names alone is unwieldy, so DOS allows you to group the files into groups called directories. Directories are similar to file cabinets; each one can hold many files, and you can add and remove them as you need. However, unlike file cabinets, a directory can contain subdirectories as well as files.

In order to do manage this system, every disk is given a single directory, named the "root" directory, when the disk is formatted. (Floppy disks usually have to be formatted before you can use them, while hard disks are usually formatted for you by the manufacturer or dealer). The root directory is capable of storing two types of structures: file names, and subdirectories. Subdirectories, in turn, are also capable of storing file names, as well as more subdirectories. This leads to a tree-like structure of directories, each of which may contain files and/or subdirectories. The term directory is used to refer to any type of directory: the root directory, and/or any subdirectory, or sub-subdirectory.

The directory structure can be easily viewed in the DOS 5.0 Shell, in the Directory Tree area of the screen. The directory structure for a relatively small hard drive is shown in Figure 5-2.



```
[-] C:\
 ├─[ ] BACKUP
 ├─[ ] DOS
 ├─[ ] DV
 ├─[-] WP
 │  ├─[ ] LETTERS
 │  │  ├─[ ] PERSONAL
 │  │  ├─[ ] BUSINESS
 │  │  ├─[ ] PR
 │  ├─[ ] BILLS
 │  ├─[ ] FLYERS
 │  └─[ ] STRATEGY
 └─[-] GV
    ├─[ ] DATA
    ├─[ ] SAMPLES
    └─[ ] TEMPLATE
```

Figure 5-2 — Portion of the Directory Tree on a Hard Disk.

The root directory for a disk is specified by a backslash (\ ). It is often preceded by the drive letter, followed by a colon. Thus, in Figure 5-2, the root directory, shown at the top, is C:\ .

All directories other than the root are assigned names. Directory names must meet the same criteria as file names: they consist of a required name, up to eight characters long, followed by an optional extension, up to three characters long. If an extension is given, it is separated from the name by a period. However, it is an almost universal convention to not use extensions for directory names. The legal characters for directory names are the same as those for file names (see Table 5-1).

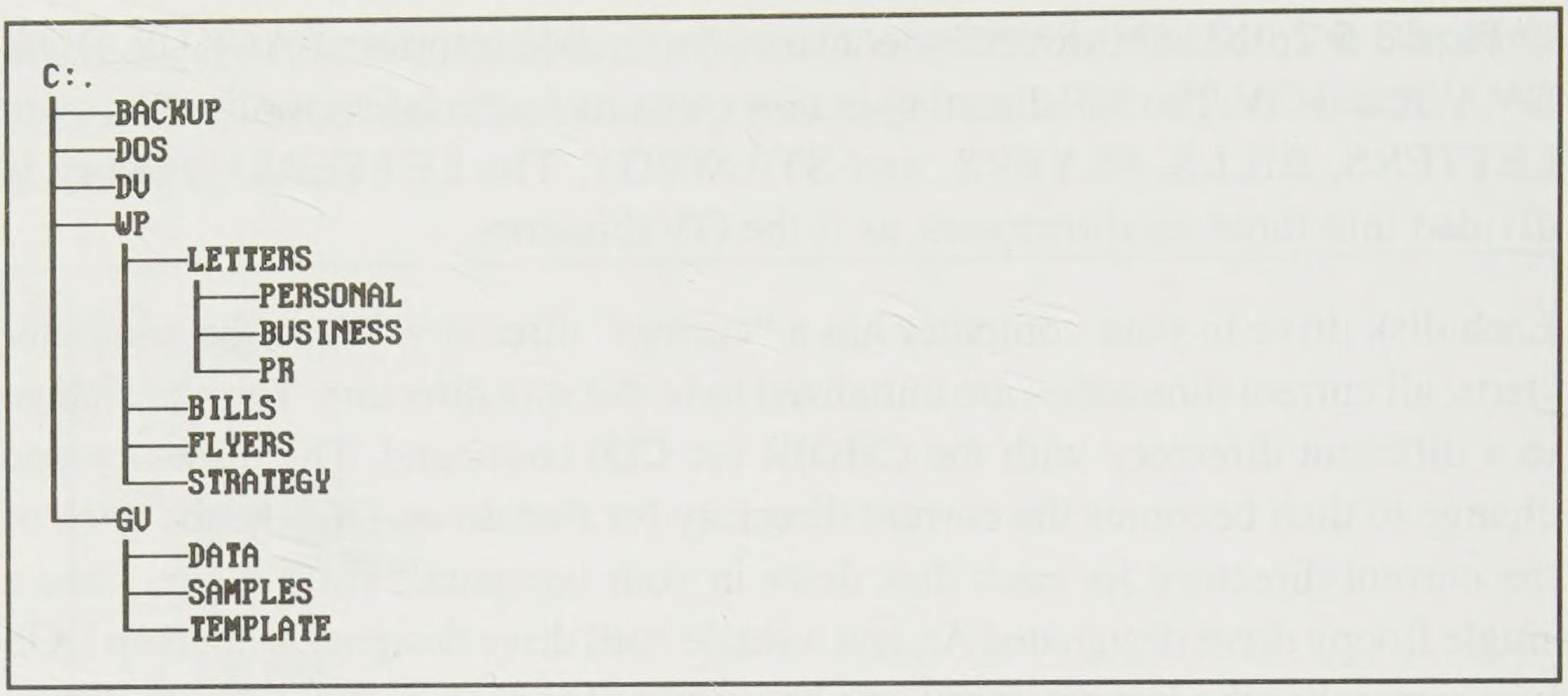
In Figure 5-2, the root directory contains five subdirectories: BACKUP, DOS, DV, WP, and GV. The WP directory in turn contains four subdirectories of its own: LETTERS, BILLS, FLYERS, and STRATEGY. The LETTERS directory is divided into three subdirectories, as is the GV directory.

Each disk drive in your computer has a "current" directory. When the computer starts, all current directories are initialized to be the root directory. You can change to a different directory with the CHDIR (or CD) command. The directory you change to then becomes the current directory for that drive. DOS keeps track of the current directory for each disk drive in your computer. Thus, if you have a single floppy drive designated A:, and a single hard drive designated C:, then DOS will remember the last directory you changed to for each of those drives. When the drive letter is referred to with no directory specification, DOS automatically goes to the current directory on that drive.

All of the files within a directory must have unique names, otherwise DOS would have no way of telling them apart. However, many files may have the same name, as long as each one is in a separate directory.

The TREE Command

The DOS 5.0 Shell provides a graphical representation of the directory tree — Figure 5-2 was generated using the Shell. However, it is also possible to examine the directory tree from the command line, using the TREE command. When entered by itself, with no parameters or switches, the TREE command displays the directory structure that begins with the current directory. Thus, using the same hard drive as in Figure 5-2, the TREE command executed from the root directory displays the structure shown in Figure 5-3.



```
C:.
├──BACKUP
├──DOS
├──DV
├──WP
│   ├──LETTERS
│   │   ├──PERSONAL
│   │   ├──BUSINESS
│   │   └──PR
│   ├──BILLS
│   ├──FLYERS
│   └──STRATEGY
└──GV
    ├──DATA
    ├──SAMPLES
    └──TEMPLATE
```

Figure 5-3 — Results of TREE Command on Root Directory.

Note that the root directory shown in Figure 5-3 is symbolized by a single period, rather than a backslash. The single period is used by DOS to refer to the current directory.

Had the TREE command in Figure 5-3 been entered when the WP directory was current, only the subdirectories that stem from WP would have been displayed — the LETTERS, PERSONAL, BUSINESS, PR, BILLS, FLYERS, and STRATEGY would have been shown. The TREE command can also be used to show the directory structure that stems from a directory other than the current one, using the conventions discussed later in this chapter, in the section titled, "Specifying Subdirectories."

There are two switches that may be used with the TREE command: /F and /A. The /F switch displays file names, as well as subdirectories. The /A switch forces the use of text characters only to draw the directory tree, rather than the graphic characters shown in Figure 5-3. If you are routing the directory tree to a printer that does not properly print the graphic characters — the lines that connect the directories — then try using the /A switch. The directory tree isn't quite as pretty, but it is readable. The /A switch may also be useful if your screen does not properly display the graphic symbols, but this is a much less frequent occurrence. The /F parameter is particularly useful for performing cleanup operations on a disk. Using the /F switch, you can print the entire contents of the disk, then go through it and see which files can be deleted. This helps to eliminate old and unnecessary files that have been forgotten, particularly on PCs that are shared by several people or that have been passed from one person to another.

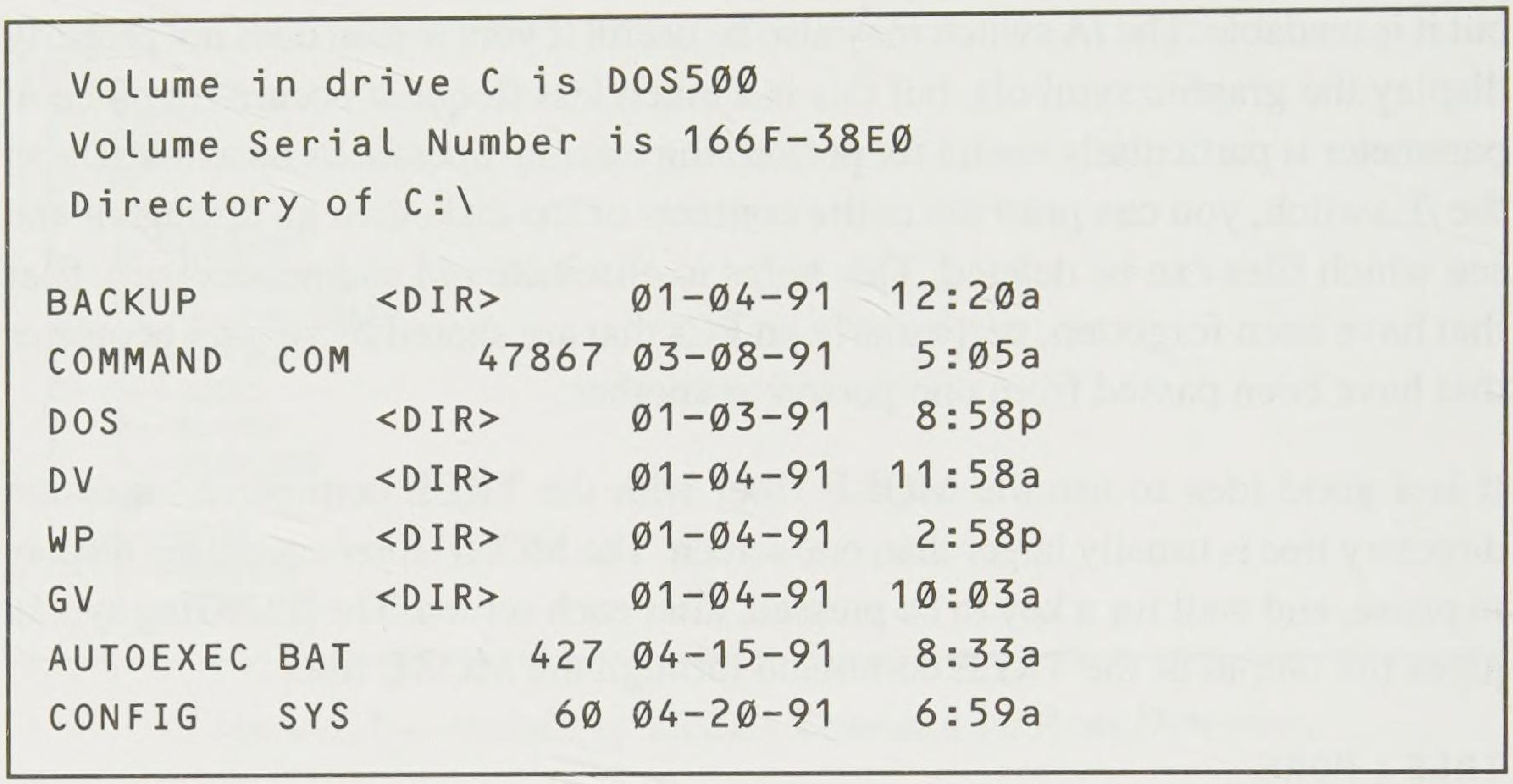
It is a good idea to use the MORE filter with the TREE command, since the directory tree is usually larger than one screen. The MORE filter causes the display to pause, and wait for a key to be pressed, after each screen. The following syntax pipes the output of the TREE command through the MORE filter:

```
TREE | MORE
```

Any of the TREE parameters and switches may be used with the MORE filter; simply place them after the TREE command, as you would without the filter, then append `| MORE` to the end of the command.

The DIR Command

The TREE command is useful when you want to get an overall picture of how the directory structure of disk is laid out, but it is a clumsy way to locate a particular file, or to see the files that are stored in a particular directory. For these purposes, the DIR command is used. When entered by itself, with no parameters or switches, DIR displays all of the files and subdirectories in the current directory. However, unlike the TREE command, it does not show the subdirectories of subdirectories; it only shows the contents of the current directory. Using the same disk as in the earlier figures, the DIR command executed from the root directory will display the output shown in Figure 5-4.



```
 Volume in drive C is DOS500
 Volume Serial Number is 166F-38E0
 Directory of C:\

BACKUP       <DIR>      01-04-91  12:20a
COMMAND  COM      47867 03-08-91   5:05a
DOS          <DIR>      01-03-91   8:58p
DV           <DIR>      01-04-91  11:58a
WP           <DIR>      01-04-91   2:58p
GV           <DIR>      01-04-91  10:03a
AUTOEXEC BAT        427 04-15-91   8:33a
CONFIG   SYS         60 04-20-91   6:59a
```

Figure 5-4 — Sample Output of DIR Command.

In addition to displaying the file and directory names that exist in a directory, the DIR command provides several other pieces of useful information.

Every disk can be given a label — a name that identifies it. The FORMAT and LABEL commands are used to assign labels to disks; these commands are fully discussed in Chapter 6. In Figure 5-4, the label is DOS500. In addition, disks formatted with DOS 4.00 or later have a serial number, which is also shown, followed by the drive and directory being displayed. You cannot change the serial number.

When a file or subdirectory is created, the date and time are recorded along with the name. Each time a file is modified, the date and time are updated to reflect the time of modification. For each file, the date and time of last modification is shown. For each subdirectory, the date and time of creation is shown. Because of its general usefulness for displaying the location and vital statistics about a directory or file, DIR is probably the most frequently used of all DOS commands.

Within the directory listing generated by the DIR command, subdirectories are identified with the symbol <DIR>.

DIR accepts a single, optional, parameter: a drive, directory, and/or file name specifier. This parameter may be used to specify a directory other than the current one, or to limit the files that are displayed to some subset of the entire directory. The following two commands demonstrate this. The first lists all files in the current directory that begin with the letter A, while the second lists all files on drive A: that have an extension of BAT:

```
DIR A*.*
DIR B:*.BAT
```

SHORTCUT

There is a shorthand technique for using the .* wild card specification with the DIR command: simply type the name part of the specifier, then press Enter←. DIR then assumes .* for the extension specifier. Thus, if the current directory contains the files REPORTS, REPORTS.JAN, REPORTS.FEB, REPORTS.MAR, and REPORTS.APR, the following command would list all of them:

```
DIR REPORTS
```

The only time this does not work is when there is a subdirectory of named REPORTS. In this case, the above command would display the contents of that directory, rather than those files that have a name of REPORTS.

In addition to the file name parameter, the DIR command accepts a number of switches (/P, /W, /A, /O, /N, /S).

These switches are shown in Table 5-3.

Table 5-3 — Switches for the DIR Command.

Switch	Effect
/A	Displays all files, regardless of attribute settings
/AA	Displays all files with the Archive attribute on
/A-A	Displays all files with the Archive attribute off
/AD	Displays directory names only
/A-D	Displays file names only - no directories
/AH	Displays all files with the Hidden attribute on
/A-H	Displays all files with the Hidden attribute off
/AR	Displays all files with the Read-only attribute on
/A-R	Displays all files with the Read-only attribute off
/AS	Displays all files with the System attribute on
/A-S	Displays all files with the System attribute off
/B	Displays brief listing - no header or summary information
/L	Displays file and directory and names in lowercase
/O	Displays directory names first, then file names; each group in sorted order
/OD	Displays files in date and time order, earliest first
/O-D	Displays files in date and time order, latest first
/OE	Displays files in alphabetic order by extension
/O-E	Displays files in reverse alphabetic order by extension
/OG	Displays directories grouped together, followed by file names, with each group in alphabetic order by name

continued…

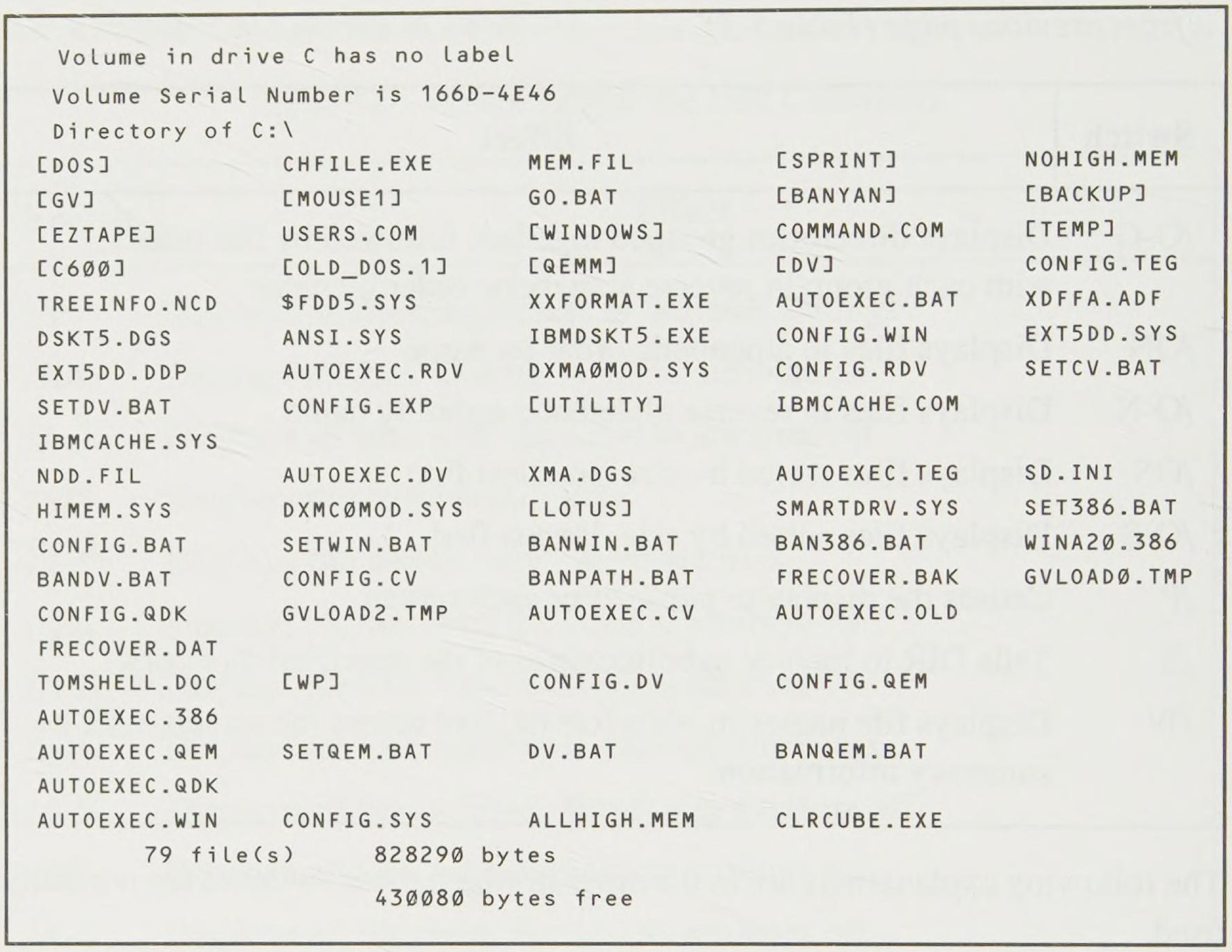
…from previous page (Table 5-3)

Switch	Effect
/O-G	Displays directories grouped together, followed by file names, with each group in reverse alphabetic order by name
/ON	Displays files in alphabetic order by name
/O-N	Displays files in reverse alphabetic order by name
/OS	Displays files sorted by size, smallest first
/O-S	Displays files sorted by size, largest first
/P	Causes the display to pause after each screen
/S	Tells DIR to include subdirectories of the specified directory
/W	Displays file names in wide format, five across the screen, with no summary information

The following explanations are in the order in which these switches are normally used.

The **/P** switch causes the display to pause after each screen, and wait for a key to be pressed. The message "Press any key to continue …" will be shown at the bottom of the screen when there is more output to come. This has the same effect as the MORE filter, but is easier to type.

The **/W** switch causes the list of file names to appear in "wide" format: the names appear in five columns across the screen. Only the names are shown — the time, date, and file size are not shown. Directories are indicated by enclosing the name in square brackets. This switch is useful for seeing a large number of file names on one screen. A listing of the root directory of a hard disk, using the /W switch, is shown in Figure 5-5.



```
 Volume in drive C has no label
 Volume Serial Number is 166D-4E46
 Directory of C:\
[DOS]           CHFILE.EXE      MEM.FIL         [SPRINT]        NOHIGH.MEM
[GV]            [MOUSE1]        GO.BAT          [BANYAN]        [BACKUP]
[EZTAPE]        USERS.COM       [WINDOWS]       COMMAND.COM     [TEMP]
[C600]          [OLD_DOS.1]     [QEMM]          [DV]            CONFIG.TEG
TREEINFO.NCD    $FDD5.SYS       XXFORMAT.EXE    AUTOEXEC.BAT    XDFFA.ADF
DSKT5.DGS       ANSI.SYS        IBMDSKT5.EXE    CONFIG.WIN      EXT5DD.SYS
EXT5DD.DDP      AUTOEXEC.RDV    DXMA0MOD.SYS    CONFIG.RDV      SETCV.BAT
SETDV.BAT       CONFIG.EXP      [UTILITY]       IBMCACHE.COM
IBMCACHE.SYS
NDD.FIL         AUTOEXEC.DV     XMA.DGS         AUTOEXEC.TEG    SD.INI
HIMEM.SYS       DXMC0MOD.SYS    [LOTUS]         SMARTDRV.SYS    SET386.BAT
CONFIG.BAT      SETWIN.BAT      BANWIN.BAT      BAN386.BAT      WINA20.386
BANDV.BAT       CONFIG.CV       BANPATH.BAT     FRECOVER.BAK    GVLOAD0.TMP
CONFIG.QDK      GVLOAD2.TMP     AUTOEXEC.CV     AUTOEXEC.OLD
FRECOVER.DAT
TOMSHELL.DOC    [WP]            CONFIG.DV       CONFIG.QEM
AUTOEXEC.386
AUTOEXEC.QEM    SETQEM.BAT      DV.BAT          BANQEM.BAT
AUTOEXEC.QDK
AUTOEXEC.WIN    CONFIG.SYS      ALLHIGH.MEM     CLRCUBE.EXE
       79 file(s)     828290 bytes
                      430080 bytes free
```

Figure 5-5 — Directory Listing Created with the /W Switch.

The /**A** switch displays only those files that have particular attribute settings. File attributes are discussed in the section titled, “How Files are Stored,” in the Advanced Topics section of this chapter.

The /**O** switch is used to display file and directory names in a sorted order. If specified by itself, the /**O** switch causes the directory list to be displayed with directory names listed first, in alphabetic order, followed by file names, in alphabetic order. However, there are several specifiers that can be appended to the switch, to designate a different sort order. Preceding the specifier with a minus sign reverses the sort order. For example, the N specifier is used to sort the list by name. The first of the following two commands sorts the list from A to Z, and the second sorts it from Z to A:

```
DIR /ON
DIR /O-N
```

An optional colon may be used between the /O and the sort specifier. This makes the command more readable, but has no real effect. Thus, the above commands above could be entered as:

```
DIR /O:N
DIR/O:-N
```

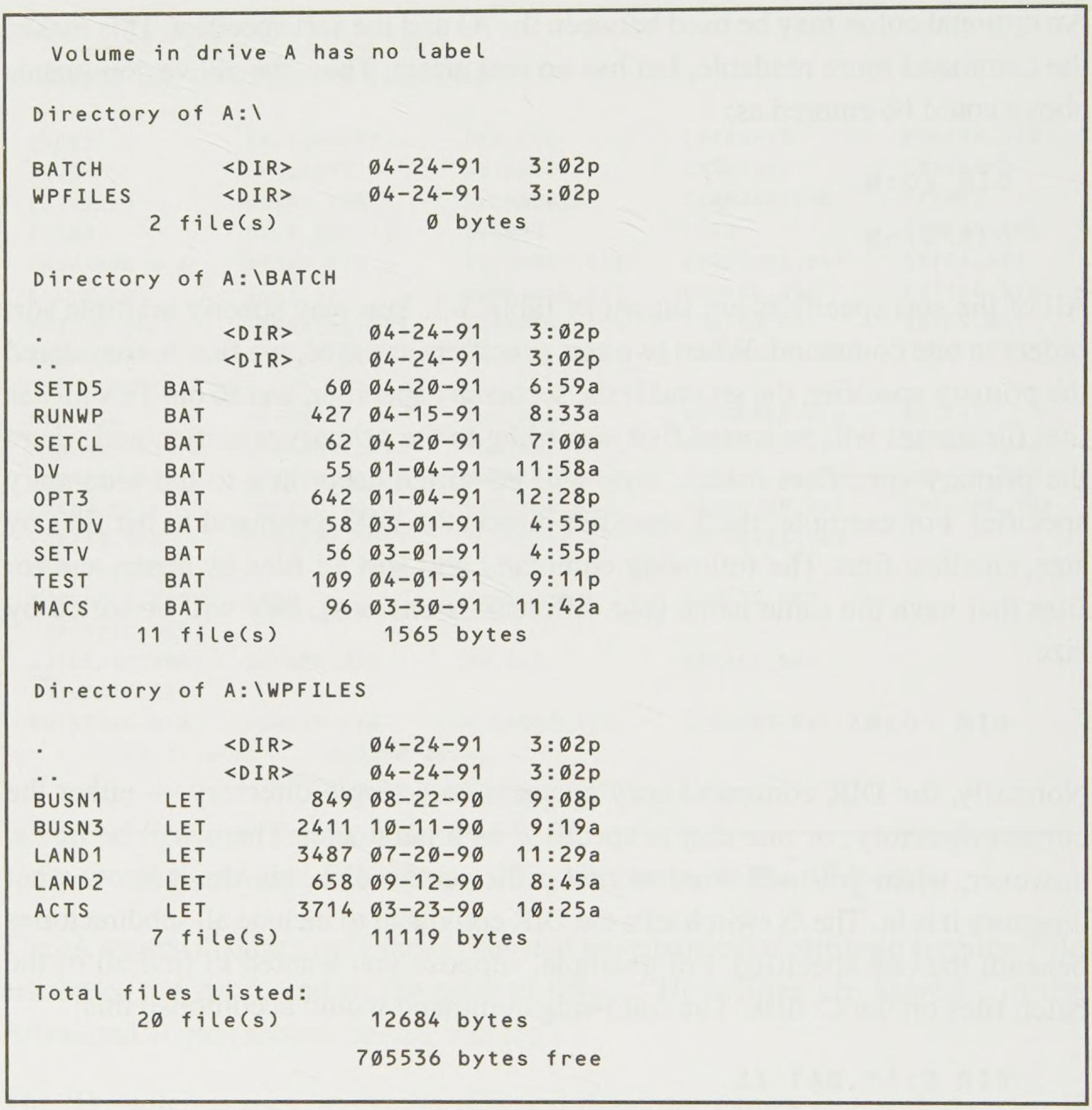
All of the sort specifiers are shown in Table 5-3. You may specify multiple sort orders in one command. When two sort specifiers are used, the first is considered the primary specifier, the second is the secondary specifier, and so on. This means that file names will be sorted first according to the primary specifier, and where the primary specifiers match, they will be sorted according to the secondary specifier. For example, the S specifier directs the DIR command to list files by size, smallest first. The following command will sort all files by name, and for files that have the same name (but different extensions), they will be sorted by size.

```
DIR /O:NS
```

Normally, the DIR command only operates on a single directory — either the current directory, or one that is specified as a parameter. There will be times, however, when you will want to find a file on the disk, but don't know what directory it is in. The /**S** switch tells the DIR command to include all subdirectories beneath the one specified. For example, suppose you wanted to find all of the batch files on the C: disk. The following command would accomplish this:

```
DIR C:\*.BAT /S
```

When /S is specified, whenever a directory has one or more files that meet the search criteria — in this example, any directory that has a file with a BAT extension — the directory name is displayed, followed by all of the selected file names. The total number of files, and the total number of bytes in those files, is shown for each directory. When all directories have been displayed, the total number of files found across all directories, and the total number of bytes in those files, is displayed. An example of such a listing, for a floppy disk, is shown in Figure 5-6.



```
 Volume in drive A has no label

 Directory of A:\

BATCH        <DIR>     04-24-91   3:02p
WPFILES      <DIR>     04-24-91   3:02p
        2 file(s)          0 bytes

 Directory of A:\BATCH

.            <DIR>     04-24-91   3:02p
..           <DIR>     04-24-91   3:02p
SETD5    BAT        60 04-20-91   6:59a
RUNWP    BAT       427 04-15-91   8:33a
SETCL    BAT        62 04-20-91   7:00a
DV       BAT        55 01-04-91  11:58a
OPT3     BAT       642 01-04-91  12:28p
SETDV    BAT        58 03-01-91   4:55p
SETV     BAT        56 03-01-91   4:55p
TEST     BAT       109 04-01-91   9:11p
MACS     BAT        96 03-30-91  11:42a
       11 file(s)       1565 bytes

 Directory of A:\WPFILES

.            <DIR>     04-24-91   3:02p
..           <DIR>     04-24-91   3:02p
BUSN1    LET       849 08-22-90   9:08p
BUSN3    LET      2411 10-11-90   9:19a
LAND1    LET      3487 07-20-90  11:29a
LAND2    LET       658 09-12-90   8:45a
ACTS     LET      3714 03-23-90  10:25a
        7 file(s)      11119 bytes

 Total files listed:
       20 file(s)      12684 bytes
                      705536 bytes free
```

Figure 5-6 — Directory Listing with the /S Switch.

The /**B** switch is used to get a Brief directory listing. A brief listing shows only file names and directory names. There is no heading for the disk label, serial number, and directory name. Neither are the file sizes, or dates and times of creation for files and directories displayed. Directories do not have any distinguishing symbols in this list, so there is no way to tell whether a name in the list applies to a file or a directory. The /B switch is useful when you want to save a set of file names in a file, for use with a program. For example, there are programs used to transfer files between different computers. Typically, these programs require just the file name to be specified; if there is extraneous information, they won't know how to handle it. To save a list of file names in a file, use the DOS output redirection symbol (>). The following command sends all of the file names in the current directory that have a BAT extension to a file named BATCH.FIL:

```
DIR *.BAT > BATCH.FIL
```

By default, the DIR command displays all file and directory names in uppercase. To display them in lowercase, the /L switch is used:

```
DIR /L
```

This is useful if you want to include such a list in a document you are preparing, and prefer it to be in lowercase, or you are preparing a file for use with a program, and the program requires lowercase file names.

It has become conventional among software developers, as well as PC users, to include extensions in the names of all files, but to never use extensions in the names of directories. Consequently, you can see the names of all subdirectories in a directory with the following command:

```
DIR *.
```

This command tells DIR to display all those items in the current directory that have nothing after the period — those files with no extensions, which is generally just the subdirectories.

Changing the DIR Default Switches

The default settings for the DIR command are:

```
DIR /A-H /A-S
```

This command has the same effect as entering DIR by itself, with file and directory names in the current directory are shown in unsorted order, and hidden and system files are not displayed. (Hidden and system files are discussed in the Advanced Topics section of this chapter). You may wish to have a different set of default switch settings and this can be accomplished with the DIRCMD environment variable. Environment variables are discussed in detail in Chapter 9. What is important to understand here is that to set the default DIR switches, the following command should be entered at the command prompt:

```
SET DIRCMD=parameter switches
```

The effect of this command is to append whatever appears after the equal sign to the word DIR, whenever the DIR command is executed. For example, if you entered the command

```
SET DIRCMD=/P
```

then whenever the DIR command was entered, the effect would be as if DIR /P was entered. You may still enter a file name or directory parameter because switches on the DIR command may be entered on either side of the parameter.

To override a preset switch, specify the same switch preceded by a minus sign on the DIR command line. For example, if the DIRCMD contained the /W switch, and you wanted to override that switch, you could use the following command:

```
DIR /-W
```

If you always want a particular set of default switch settings, the `SET DIRCMD=` command can be included in your AUTOEXEC.BAT file.

The DIRCMD environment variable may be changed at any time, by issuing a new `SET DIRCMD=` command. These commands only stay in effect until you turn off the computer. The next time the computer is booted, you will have to re-enter the SET command. To see what the current setting of the DIRCMD variable is, enter the command SET at the command prompt. A list of any environment variables that have been set will be shown. If the DIRCMD environment variable has been set, it will be among them.

You may combine as many switch settings as you like on one command line. For example, the following command lists all of the files in the current directory, sorted by extension, in wide format, and excludes directory names from the list:

```
DIR /OE /W /A-D
```

The DIR command can also be used to show the file and subdirectory contents of a directory other than the current one, using the conventions discussed in the next section.

Specifying Subdirectories

DOS keeps track of whatever directory you are currently working in (the current directory). When a file name is used as a parameter to a command, DOS looks only in the current directory to locate it. If the file cannot be found there, DOS displays an error message. However, since it is often necessary to refer to files that are not in the current directory, DOS provides several convenient symbols for specifying the directory in which a file is located along with the file name.

Since the root directory does not have a name, the backslash character is used to identify it. Thus, to see the directory listing for a file named ROBERT.LET that is stored in the root directory, the following command could be used:

```
DIR \ROBERT.LET
```

If the root directory were the current directory, the backslash would not be necessary, but if any other directory were current, then this command would still display the directory listing of the ROBERT.LET file in the root directory.

The backslash is also used to separate subdirectory names and file names when specifying the location of a subdirectory or file. For example, to display all of the files in the LETTERS subdirectory of the WP directory, on the C: drive, the following command is used:

```
DIR C:\WP\LETTERS
```

In addition to the backslash, there are two more special symbols used in specifying directories: the single period (.) and the double period (..). These two symbols are used by DOS to keep track of the directory tree. The single period refers to the current directory. Thus, the command

```
DIR .
```

is the same as the command

```
DIR
```

The double period refers to the parent directory — the directory for which the current directory is a subdirectory. All directories have a parent directory, except for the root directory. The double period makes it easy to specify directories that are nearby. For example, referring to the directory tree in Figure 5-3, if the current directory is C: \ WP \ LETTERS \ BUSINESS, then the following command will display a listing of all files in the C: \ WP \ LETTERS directory:

```
DIR ..
```

Again, if the current directory is C: \ WP \ LETTERS \ BUSINESS, then the following command displays a listing of all files with the extension LET, in the PR directory:

```
DIR ..\PR\*.LET
```

These abbreviations are much shorter and easier to type than the full path name for the specified directories.

Printing and Saving Directory Listings

You may send a directory listing to the printer by using the output redirection symbol (>) with the PRN specifier, as follows:

```
DIR > PRN
```

You may also use the output redirection symbol to store a directory listing in a file. Simply replace the word PRN above with the name of a file. For example, to store a listing of all files in the C: \ WP \ LETTERS directory in a file named LIST.DIR, in the C: \ GV \ DATA directory, the following command would be used:

```
DIR C:\WP\LETTERS > C:\GV\DATA\LIST.DIR
```

Organizing Directories

The DOS directory system provides an excellent means of organizing and arranging your files. However, if there is no method as to how it is used, you could end up with more chaos than if you simply put all of your files into the root directory. How your directories should be organized is, therefore, worth a lot of careful thought.

There are basically two approaches to organizing directories: one is to have one set of directories for programs, and another for data files. For example, suppose you have three programs you want to use on your computer: WordPerfect for word processing; Excel for spreadsheet work; and R:BASE for database management. You could create one subdirectory beneath the root for each of these programs, then another subdirectory beneath the root for the data files that are used with each program. The data file subdirectories could be further divided into more subdirectories, if you had a variety of categories for the data files for each program.

The second approach is to create a subdirectory for each program, then create subdirectories beneath it for the data files that go with the program. In this scheme, following the example above, you would have three subdirectories beneath the root: WordPerfect, Excel, and R:BASE. Beneath each of these would be one or more subdirectories for the data files that go with the program.

Either system works well. What is important is to decide on an approach, and stick to it. You should understand where in your system a particular file or directory should fit, before you create it. Creating directories without having an overall plan for your directory structure will lead to a cluttered disk and files that are difficult to locate.

DOS Commands for File and Directory Management

Nearly one third of all DOS commands are used for file and directory management. This reflects the central role that files and directories play in using a Personal Computer, and the importance of having commands to facilitate managing them. This section arranges these commands into logical groups. Within groups, they are presented alphabetically, except where two related commands should logically follow each other. Each command is identified as being either an internal or external command. Internal commands are commands that DOS stores in memory at the time the computer is booted. Therefore, they do not have to be read in from the disk before they can be used. External commands, on the other hand, are commands that DOS does not store in memory. Consequently, they must be read from the disk, just like any other program, before they can be used.

Commands for Comparing, Copying, and Deleting Files

The **COMP** command compares the contents of two files. The files are compared on a byte-by-byte basis. Since a byte is the smallest unit that can be stored in a file, COMP performs a precise comparison of the two files. The files can be in the same or different drives and directories. The syntax for the COMP command is:

```
COMP filespec1 filespec2 switches
```

The file specifiers may include drive and/or directory references, as well as file names. Wild cards are permitted, so groups of files may be compared. For example, the following command compares all batch files in the A:\ directory with all batch files in the C:\ directory:

```
COMP A:\*.BAT C:\*.BAT
```

You do not have to provide a complete file specifier for either filespec1 or filespec2. If you enter the command COMP with no file specifier, then you are prompted for the first file specifier, then for the second file specifier, and finally for any optional switches. If you enter only a drive specifier, or only a directory specifier, for filespec1, then COMP assumes that the file specifier is *.*. If filespec2 contains only a drive or directory specifier, then COMP assumes the file name in that drive or directory will be the same as that in filespec1. Thus, the following command takes each file in the A: drive root directory, and looks for a file of the same name in the C: drive root directory. If a matching file name is found, the files are compared:

```
COMP A:\ C:\
```

If no matching file name is found in the second directory, COMP displays the message "Can't find/open file: filespec2", where filespec2 is the file name for the second file. It then goes on to the next file in filespec1.

When COMP compares two files, it displays the message "Files compare OK" if they match. If they do not match, COMP displays a message indicating the exact location in each file at which the mismatch occurred. To demonstrate, the following two files were compared:

```
filespec1 (named TEST1.FIL):

This is line one.

Line two is here.

filespec2 (named TEST2.FIL):

This is line one.

Fine two is here.
```

The difference between these files is that the first character of the second line is the letter L in the first file, and the letter F in the second file. The COMP command displays these results:

```
Comparing TEST1.FIL and TEST2.FIL...

Compare error at OFFSET 13

filespec1 = 4C

filespec2 = 46
```

The position of the error is shown in hexadecimal; the hex value 13 is the decimal number 19. Hence, there is a mismatch in the two files at position number 19. There is a carriage return symbol at the end of the first line, which is why the first character of line two is at position 19, rather than 18. COMP also says what the difference is — the first file contains the number 4C (hex), and the second file contains the number 46 (hex). These numbers, when translated through the ASCII table, correspond to the letters L and F.

There are five switches that COMP accepts, all of them are optional. The /A switch causes mismatches to be displayed as characters rather than as numbers. When the above two files are compared with the /A switch, the following output appears:

```
Comparing TEST1.FIL and TEST2.FIL...

Compare error at OFFSET 13

filespec1 = L

filespec2 = F
```

Normally, the comparison that COMP makes is case sensitive: it will detect a mismatch if one file contains, say, a lowercase "a" in the same position as the other file has an uppercase "A." The /C switch turns off case sensitivity, so that upper and lowercase versions of the same letter are seen as identical.

For many people, it is preferable to see the differences between the files displayed as decimal numbers, rather than hexadecimal. This can be specified with the /D switch. The same comparison as shown above, this time with the /D switch, produces the following output:

```
Comparing TEST1.FIL and TEST2.FIL...

Compare error at OFFSET 13

filespec1 = 76

filespec2 = 70
```

The different characters are now shown to be ASCII codes 70 and 76 in decimal. However, the offset value is still specified in hexadecimal.

It is often more useful to know the line number on which a mismatch occurs, rather than the number of bytes into the file. This is specified with the /L switch. In this case, the output for the example would say "Compare error at LINE 2" instead of "Compare error at OFFSET 13." When the /L switch is used, the line number is displayed in decimal, not hexadecimal, format.

The last switch is /N=number, where number is the number of lines to compare in the files. Without this switch, if the two files do not have exactly the same size, COMP will refuse to compare them, instead displaying the message "Files are different sizes." The /N parameter tells COMP to ignore the file sizes, and compare the specified number of lines. Thus, if one file is shorter than the other, you can still determine if the two are identical up to the end of the shorter one.

For most purposes, the most useful form of the COMP command is:

```
COMP filespec1 filespec2 /A /L
```

This shows the location of the mismatch as a line number, and it displays the mismatched characters as letters rather than ASCII codes.

If COMP encounters more than 10 mismatches, it displays the message "10 mismatches - ending compare," and terminates the file comparison. Whenever COMP finishes — whether the files matched or did not — it asks if you want to compare more files. If you enter a Y, it then prompts you for the first and second file names, and any optional switches. If you don't enter any switches, COMP uses the same set of switches entered when you started the command. If you enter an N, it returns you to the command prompt.

You may also want to review the FC command, discussed below, for comparing files.

COMP is an external command.

The **COPY** command copies a file from one location to another. The second location may be anywhere that a file could reside — another disk, another directory on the same disk, or even the same directory on the same disk, if the name of the source file and target file are different. The simplest form of the COPY command is:

```
COPY file1 file2
```

This command copies a file named file1, in the current directory, to a file named file2, also in the current directory. In this case, file2 cannot have the same name as file1, since you cannot have two files with the same name in the same directory. If you attempt to copy a file to the same name in the same directory, DOS will display the following message:

```
File cannot be copied onto itself
    0 file(s) copied
```

However, if you include a drive and/or directory specifier for either or both files, then the second file may have the same name as the first, as long as the drive and/or directory is different. In fact, you do not even have to specify a file name for the second file. If no file name is specified for file2, the COPY command assumes it should have the same name as file1. For example, assuming that the current drive and directory are C:\WP\LETTERS, the following four commands all perform exactly the same function:

```
COPY A:\ROBERT.LET C:\WP\LETTERS\ROBERT.LET
COPY A:\ROBERT.LET C:\WP\LETTERS
COPY A:\ROBERT.LET C:
COPY A:\ROBERT.LET
```

Each of these commands copies a file named ROBERT.LET from the root directory of the A: drive to a file named ROBERT.LET in the current directory, which is C:\WP\LETTERS. The first command provides a complete specification for both source and target files. The second provides a drive and directory specification for the target file, but leaves it up to the COPY command to use the same target file name as the source. The third command lets COPY retrieve the current directory on the C: drive from DOS. As long as only the drive letter is specified, with no directory symbols or names, then the current directory for that drive is assumed. The fourth command lets COPY assume the entire current drive and directory from DOS. If you want to use a different name for the target, then that name must be specified. However, you may still let COPY assume a current drive and/or directory.

DOS wild card symbols may be used to copy sets of files. For example, to copy all files with an extension of BAT from the root directory into a directory named BATCH, enter the following command:

```
COPY \*.BAT \BATCH
```

When the COPY command is used to duplicate files, the new files have the same date and time stamps as the original files, since they have not been modified since that date and time. However, the COPY command may also be used to combine files, by using a plus sign between the names of the files to be combined. When a combination is performed, the target file is modified, and so the date and time stamp of the target are set to the current date and time. The following command combines two files, named TEST1.FIL and TEST2.FIL, into a third file, named TEST3.FIL:

```
COPY TEST1.FIL + TEST2.FIL TEST3.FIL
```

If no target file is specified when combining files, the first file named is assumed to be the target. For example, the following command combines TEST1.FIL and TEST2.FIL, and stores the result back in TEST1.FIL:

```
COPY TEST1.FIL + TEST2.FIL
```

You may also use wild cards to combine files. For example, the following command combines all batch files into a single file named BATCH.ALL:

```
COPY *.BAT BATCH.ALL
```

Wild cards may also be used to selectively combine one set of files with another set, resulting in a third set. For example, the following command combines all files with the extension LET, with all the corresponding files of the same name but with the extension ENV, producing files with the same name but the extension FIN:

```
COPY *.LET + *.ENV *.FIN
```

Thus, the file ROBERT.LET is combined with the file ROBERT.ENV, producing the file ROBERT.FIN. Likewise, SUSAN.LET and SUSAN.ENV are combined to produce SUSAN.FIN, and so on. On the other hand, if you wanted to combine all of the LET and ENV files into one big file named ALL.FIN, the following command could be used:

```
COPY *.LET + *.ENV ALL.FIN
```

WARNING

Caution: When combining files, if the name of the target file is the same as the name of one of the source files other than the first, then the file identified as the target is overwritten before its original contents are read. In this case, the contents of that file are not copied into the target, and DOS displays the following message when the copy operation is complete:

```
Content of destination lost before copy
```

As an example, consider the following command:

```
COPY TEST1.FIL + TEST2.FIL + TEST3.FIL TEST2.FIL
```

This command attempts to concatenate TEST2.FIL with TEST1.FIL and TEST3.FIL, and store the result in TEST2.FIL. Here is what the output of the command looks like:

```
TEST1.FIL
TEST2.FIL
Content of destination lost before copy
TEST3.FIL
        1 file(s) copied
```

The result of this command is that TEST2.FIL contains the combined contents of TEST1.FIL and TEST3.FIL, but the original contents of TEST2.FIL are lost.

The COPY command accepts three optional switches: /A, /B, and /V. The **/V** is used to verify a copy — it performs a test during a copy operation to make sure that the new file is written correctly. Copying with the /V switch takes considerably longer than copying without it, because DOS must double check everything that is written into the target file. In general, the /V switch is not needed because the

COPY command is quite accurate. However, when copying critical data under circumstances where a bad copy would create severe problems, it is worth using /V to make sure the copy is completed successfully.

DOS distinguishes between two types of files: ASCII text files, and binary files. An ASCII text file contains only ASCII codes, while a binary file contains information that is not interpreted according to the ASCII table. Some word processing files are ASCII text files, while program files and most spreadsheet and database files are binary files. For the purposes of the COPY command, the significant difference is in how the two types of files indicate where the file ends. ASCII text files indicate the end of file with a control-Z character, which is known as the "end of file" marker. The control-Z character is displayed as a caret followed by the letter Z like this: ^Z. Even though it is displayed as two characters, control-Z is actually stored as a single character (1A hex).

To determine the end of a binary file, on the other hand, the COPY command uses the file size as specified in the directory. For example, if a file's directory listing shows it as containing 1537 bytes, then COPY will copy exactly 1537 bytes, starting from the beginning of the file.

The /**A** switch tells COPY to interpret the file as an ASCII text file, and therefore determine the end of the file by looking for a control-Z. It is possible for a file to contain more than one control-Z, but with the /A switch, COPY only copies the file up to, but not including, the control-Z. However, when the entire COPY is finished, if /A was specified then COPY places a final control-Z at the end of the target file.

The /**B** switch, on the other hand, tells COPY to interpret the file as a binary file, and to determine the length of the file from its directory entry. Everything in the file is copied, including any control-Z characters. No control-Z is automatically added to the end of the target file when /B is specified.

When files are not being combined, but simply copied, then /B is the default because this technique for determining the end of the file works for either file type. If the source file has a control-Z at the end of it, the control-Z is copied. If it does not, then none is added. However, when combining files, the /A switch is the default because with this switch, ASCII files are correctly combined, with only a single control-Z ending up in the target directory. Since it is rare to combine binary files, but relatively common to combine ASCII text files, it makes sense to use /A as the default for combining files. However, if you are combining binary files, you must be sure to specify the /B switch. Otherwise, a control-Z will be added to the end of the target file, probably making it unusable.

The /A and /B switches may be combined in a single command to treat different files in different ways. When either switch precedes all of the file names, then it applies to all file names up to but not including the one that precedes the appearance of the other switch. The other switch then applies to all file names up to but not including the one that precedes the reappearance of the first switch, and so on. For example, in the following command, the initial /A applies to FILE1 and FILE2. The /B applies to FILE3 and FILE4, and the second /A applies to FILE5, FILE6, and FILE7:

```
COPY /A FILE1 + FILE2 + FILE3 /B + FILE4 + FILE5 /A + FILE6 FILE7
```

The COPY command can also be used to trick DOS into changing the date and time stamp of a file to the current date and time. The command to accomplish this is:

```
COPY /B TEST.FIL+,,
```

This command looks strange, but it has meaning. The /B tells the COPY command not to append a control-Z to the end of the file. This is important because the file should be left unchanged, and /A (the default for combining files), will add the control-Z at the end of the file. The file name is TEST.FIL, and the + sign following it says "this is a combining operation." However, the first comma says "omit the second source file," and the second comma says "omit the target file." The result is that the file is left unchanged, but since this is a file combination operation, the date and time stamp are updated.

The COPY command can also be used to copy to or from a device, such as the printer or the keyboard. As far as DOS is concerned, devices are no different from files — they are either sources or targets for transferring information. The term CON is a reserved word that DOS uses to refer to the "console" (the keyboard and the screen). When the word CON appears in a place where a source file would be specified, then input is taken from the keyboard. When the word CON appears in a place where a target file would be expected, then output goes to the screen.

Consequently, the COPY command can be used to create a file using the following command:

```
COPY CON TEST.FIL
```

This command will take the keyboard as the source file, and create a file named TEST.FIL that contains whatever is typed at the keyboard. This will be an ASCII file, since only ASCII characters can be generated by the keyboard. Recall that ASCII files are terminated with a control-Z character. To tell DOS when to stop copying from the keyboard, a control-Z character is entered by holding down Ctrl and pressing Z. A shortcut is to press F6, which generates a control-Z. Enter← must be pressed after the control-Z. In the same manner, the contents of a file may be displayed on the screen with the following command:

SHORTCUT

```
COPY TEST.FIL CON
```

This will display the contents of the file TEST.FIL on the screen. To see how these two commands work, enter the following:

```
COPY CON TEST.FIL Enter←
Hello. This is a test file Enter←
being created with the COPY command. Enter←
This is the last line of the file. Enter←
^Z Enter←
COPY TEST.FIL CON Enter←
```

The first five lines create the file TEST.FIL, and the last line displays it back on the screen.

The word PRN is another reserved word. DOS uses it to refer to a printer port. If you have a printer connected to your computer, then you can use the COPY command to print out an ASCII text file, as follows:

```
COPY TEST.FIL PRN
```

There are two types of copying operation that COPY does not perform: it cannot copy files that have a file size of 0 bytes, and it cannot copy files from more than one directory at once. Both of these operations can be accomplished with the XCOPY command, discussed below.

COPY is an internal command.

The **DEL** command deletes files from a disk. The command accepts wild cards as well as individual file names, and a drive and/or directory may be specified. To delete one or more files, enter the following command:

```
DEL filespec
```

For example, to delete all files with a BAK extension in the root directory of the C: drive, enter the following command:

```
DEL C:\*.BAK
```

DEL accepts one optional switch: /P. This switch tells DEL to prompt you for confirmation before each file it deletes. For example, if there were files named AUTOEXEC.BAK, CONFIG.BAK, and WP.BAK in the current directory, then entering the above command with the /P switch would cause DEL to display the following message:

```
C:\AUTOEXEC.BAK,    Delete (Y/N)?
```

Pressing Y deletes the file, while pressing N leaves the file intact. You do not need to press Enter← after pressing the Y or N — the keystroke is accepted as soon as you press it. The message then will be repeated for each of the other files. If you press any key other than Y or N, the prompt line will be repeated until you press a Y or N.

STOP

Be careful when you use the DEL command, particularly when using wild cards, since this can delete a large number of files at once. If you accidentally erase one or more files, they may be recoverable using the UNDELETE command, discussed later in this chapter in the section titled, “File Protection and Recovery Commands.” However, this is time consuming and only works under certain circumstances. When using wild cards, it is particularly important to specify the /P switch to make sure you do not delete a file you want to keep. An alternative is to first use the DIR command with the same wild card specification, to review all of the file names at once.

There are two ways to delete all the files in a directory: using the "many" wild card (*) for both the name and the extension specifier, and using just the directory name, with no file specifier. Thus, both of the following commands accomplish the same thing — deleting all files in the C:\WP\LETTERS directory:

```
DEL C:\WP\LETTERS\*.*

DEL C:\WP\LETTERS
```

Deleting all the files in a directory is a fairly drastic move, so DEL has built-in protection to prevent mistakes. When a request is made to delete all the files in a directory, DEL displays the following message:

```
All files in directory will be deleted!

Are you sure (Y/N)?
```

In this case, you must type either a Y or an N, and then Enter←. Enter← is required in case the wrong key is accidentally pressed.

DEL is an internal command.

Note: In previous versions of DOS, the ERASE command was also included in DOS, and was synonymous with the DEL command. Since many people have created batch files that use ERASE, and have gotten used to typing it, it continues to work in DOS 5.0, though it is not officially listed as a legal command. It is a good idea to get used to using DEL instead of ERASE, since in future versions of DOS, ERASE may no longer be supported.

The **DISKCOPY** command creates an exact duplicate of a disk. When DISKCOPY is used, the source disk is duplicated byte for byte on the target disk. Since every byte on the target disk is copied from the source disk, any information that may be stored on the target is destroyed by the DISKCOPY command. To understand how DISKCOPY is different from using the COPY or XCOPY command to copy all of files on a disk, it is necessary to understand a little about how files are stored on disk.

When files are created on a disk, space is not allocated to them on a byte-by-byte basis. Instead, space is allocated in units called clusters. The size of a cluster varies with different types of disks; a cluster of 2048 bytes is common for hard disks, and clusters of 512 and 1024 bytes are common for floppy disks. So when a file is created on a typical hard disk, DOS will reserve a minimum of 2048 bytes for it. If the file size is 2049 bytes, then two clusters (4096 bytes) will be reserved for it, and so on.

As files are created and deleted, the available clusters on the disk become non-contiguous or fragmented. Imagine creating four files on a new disk: file1, which occupies one cluster; file2, which occupies three clusters; file3, which occupies two clusters; and file4, which occupies one cluster. Thus, the arrangement on the disk will be:

```
< file1 ><     file2     ><  file3      >< file4 >

1 cluster    3 clusters    2 clusters     1 cluster
```

Beyond file4 is a large amount of empty disk space. Now suppose file3 is deleted. That leaves a two-cluster hole in the file sequence. The free disk space is now fragmented into two pieces: the two-cluster hole between file2 and file4, and the large space beyond file4. If file5 is now created, and requires four clusters, where will it be placed? It won't fit into the two cluster hole, but if files could only be put into contiguous clusters, eventually the disk would become so fragmented that much of it would be unusable. So DOS puts the first two clusters of file5 into the hole where file3 used to exist, and the rest of it into the space following file4. This allows all disk space to be used, regardless of the degree of fragmentation. Fragmentation is discussed in detail in Chapter 6.

Now that you understand how files are stored on the disk, you can understand the main difference between DISKCOPY and COPY. When a file is copied from one disk to another with the COPY command, the copy of the file is placed into whatever clusters are free on the target disk. Thus, the two copies of the file may be in a totally different areas of the two disks. When DISKCOPY is used, the second disk is an exact byte-for-byte duplicate of the first — every file occupies exactly the same clusters on both disks.

Since DISKCOPY must make an exact duplicate of a disk, it makes sense that only disks that are the same size and format can be used with DISKCOPY. Furthermore, DISKCOPY only works with floppy disks. Even if your computer has two hard disks of the same size, DISKCOPY cannot be used to duplicate one on the other. The XCOPY command was designed for this purpose. You rarely need to make an exact duplicate of a hard drive, but it is sometimes necessary to duplicate floppies. DISKCOPY provides a convenient mechanism for accomplishing this. Since DISKCOPY copies every byte on a disk, hidden files and system files are also copied, and their attributes left unchanged. Consequently, if the source disk is a startup disk, the target disk will be as well.

Note: Some original program disks contain data stored before or after the normal recording tracks to prevent duplication (on track –1 or track 41, on a 40-track floppy, for example). DISKCOPY will not copy this extra data.

If the target disk has not been formatted, DISKCOPY will automatically format it as it goes, and will display the message "Formatting while copying," to let you know what it is doing. This will take longer than copying to a formatted disk.

If the source and/or target drive is not specified, DISKCOPY will use the current drive for the missing drive specifier(s). Thus, DISKCOPY can be used with computers that have a single drive. When the same drive is used as the source and target, DISKCOPY alternately prompts you to place the source or target disk in the drive.

There are two optional switches that may be used with the DISKCOPY command: /1 and /V. Floppy disks can be either single sided or double sided, though single sided disks are rarely seen anymore. Normally, DISKCOPY determines the number of sides based on the type of disk drive that DOS senses. However, you can force DISKCOPY to copy only the first side of a disk with the /1 parameter (it's the numeral one, not the letter L). The /V switch is the same as with the COPY command — it causes DISKCOPY to verify that the information is copied accurately. The /V switch slows down the copy operation.

If the source disk has a volume serial number, DISKCOPY will generate a new volume serial number for the target disk. This number is displayed when the copying operation is complete.

There is a DOS environment variable named ERRORLEVEL. Some DOS commands, as well as some software programs, leave a value stored in this variable when they stop executing. This value can then be used to determine the results of running the program. The DISKCOPY command sets the ERRORLEVEL variable to a number from 0 to 4; the meaning of each value is shown in Table 5-4.

Table 5-4 — Values of ERRORLEVEL Set by DISKCOPY Command.

Value	Meaning
0	DISKCOPY was successful.
1	A nonfatal read/write error occurred.
2	DISKCOPY was terminated by the user pressing Ctrl-C or Ctrl-break
3	A fatal hard error occurred.
4	An initialization error occurred. This is caused by one of the following problems: insufficient memory; insufficient disk space; an invalid drive specifier; or a syntax error in the DISKCOPY command.

The ERRORLEVEL variable can be referenced on the IF line of a batch file, and by programs that are designed to check for it to take an appropriate action if DISKCOPY returns a value other than 0. Batch files are discussed in Chapter 8.

DISKCOPY is an external command.

FC Compares two files and displays the differences between them. The difference between the FC and COMP commands is that COMP is designed primarily for determining that two files are identical, while FC is designed primarily for helping you see the differences between two files. COMP is typically used after copying a file, to see if the copy was successful. FC, on the other hand, is typically used to compare two files that are known to be different, in order to decide which one to keep.

Unless you specify otherwise with the optional switches, FC assumes the files are in ASCII format, and compares them line for line. When it encounters a mismatch, it displays the complete lines that do not match in each file, as well as the surrounding lines. This makes it easy to see the significance of the difference between the two files. The syntax of the FC command is:

```
FC file1 file2
```

In order to demonstrate the FC command, the following two files will be used:

```
TEST1.FIL
This is line one of the test file.
This is line two of test file1.
This is line three of test file1.
This is line four of the test file.
This is line five of the test file.
This is line six of the test file.
This is line seven of the test file.
This is line eight of test file1.
```

```
TEST2.FIL
This is line one of the test file.
This is line two of test file2.
This is line three of test file2.
This is line four of the test file.
This is line five of the test file.
This is line six of the test file.
This is line seven of the test file.
This is line eight of test file2.
```

The command FC TEST1.FIL TEST2.FIL produces the following output:

```
Comparing files TEST1.FIL and TEST2.FIL
    ***** TEST1.FIL
    This is line one of the test file.
    This is line two of test file1.
    This is line three of test file1.
    This is line four of the test file.
    ***** TEST2.FIL
    This is line one of the test file.
    This is line two of test file2.
    This is line three of test file2.
    This is line four of the test file.
    *****
    ***** TEST1.FIL
    This is line seven of the test file.
    This is line eight of test file1.
    ***** TEST2.FIL
    This is line seven of the test file.
    This is line eight of test file2.

    *****
```

The first thing FC does is display a line that names the two files being compared: Comparing files TEST1.FIL and TEST2.FIL. It then starts scanning each file, comparing them line by line. When it encounters a line that does not match, it displays the name of the first file, the last line that did match, all of the consecutive lines that do not match, then the first matching line that follows. FC then displays the same information for the second file. When FC changes from one file to another, it displays a line containing five asterisks.

In the example above, the first line that doesn't match is line number two (in file1, line number two ends with "file1," while in file2 it ends with "file2"). Therefore, FC displays the first line of the first file (which was the last line that matched), then displays lines two and three, both of which contain mismatches. Finally, line four is displayed, which is the first line that matches following the mismatches. This information is then repeated for the second file. FC prints a blank line before displaying the next mismatch.

The output of the FC command makes it very easy to see precisely where mismatches occur. For each group of displayed lines, the first and last lines surround the mismatched lines, and all lines in between the first and last line contain mismatches.

There are several switches that can be used with FC. The /B switch is used to compare binary, rather than ASCII, files. The difference between these two file types was previously discussed in the COPY command section. When the files being compared have a BIN, COM, EXE, LIB, OBJ, or SYS extension, this setting is assumed unless the /L switch is used. For all other file extensions, the /B switch is off by default. When /B is in effect, FC does not attempt to resynchronize the files after finding a mismatch, and mismatches are reported in the following format:

```
XXXX: YY ZZ
```

Where **XXXX** is a hexadecimal number representing the byte position within the file of the mismatch, **YY** is a hexadecimal representation of the mismatched byte in file1, and **ZZ** is a hexadecimal representation of the mismatched byte in file2. For example, the following command compares the two files shown above, using the /B switch:

```
FC TEST1.FIL TEST2.FIL /B
```

The output of this command is:

```
Comparing files TEST1.FIL and TEST2.FIL
00000041: 31 32
00000064: 31 32
0000011B: 31 32
```

The first mismatch appears at position 41h (decimal 65); in TEST1.FIL, this byte contains 31h (decimal 49, which is the ASCII symbol for 1), and in TEST2.FIL it contains 32h (decimal 50, which is the ASCII symbol for 2).

The /L switch forces the comparison into ASCII mode. For files that have any extension other than BIN, COM, EXE, LIB, OBJ, and SYS, ASCII mode is the default and /L has no effect. For files that have one of these extensions, binary mode is the default and the /L switch is needed for an ASCII comparison.

When FC compares two files in ASCII mode, it loads up to 50 lines at a time from each file, compares those lines, then loads more lines, and so, until all lines have been compared. However, if at any time FC finds no matching lines in the sections that are in memory, it displays the following message:

```
Resync failed. Files are too different.
```

The meaning of this message is that FC was unable to resynchronize the files: it could not find a common point for restarting the comparison after encountering the mismatches. When comparing files in binary mode, however, FC will compare the files byte for byte, from beginning to end, regardless of how many mismatches it encounters.

The reason that FC is limited to comparing no more than 50 lines per file in ASCII mode is that ASCII mode comparisons are done by loading the lines from each file into a buffer that FC creates in memory. This buffer can hold up to 100 lines. This number can be increased by using the /LBx switch, where **x** is the total number of lines to be stored in the buffer. For example, to double the size of the buffer, the following command would be used:

```
FC /LB200 TEST1.FIL TEST2.FIL
```

If there are more than the specified number of consecutive mismatched lines, FC will display the “Resync failed” message.

The /A switch is used to abbreviate the output when comparing in ASCII mode. Without this switch, if 10 consecutive lines are different, then FC displays all 10 lines. With /A, only the matching lines that surround the consecutive mismatching lines are displayed.

You can tell FC to display the line numbers of the mismatched lines, using the /N switch:

```
FC /N TEST1.FIL TEST2.FIL
```

When FC encounters a tab character in a file, it expands the tab to eight spaces. Tab expansion can be shut off with the /T switch. You may want to compress all "white space" — blank spaces and tab stops. This is accomplished with the /W switch. When /W is specified, white space is not entirely removed, but instead consecutive spaces and/or tabs are compressed into a single space.

Normally, FC comparisons are case sensitive, so that lowercase **a** and an uppercase **A** are seen as different. Case sensitivity is turned off with the /C switch.

When FC encounters a mismatch, it must resynchronize the files — it must decide at what point to begin comparing them again. FC considers files to be resynchronized when two consecutive lines have matched. If the number of matching lines is less than two, then it continues displaying the lines as mismatched. You may adjust the number of lines that must match by using the slash (/) symbol, followed by the desired number of lines. For example, the following command compares the files TEST1.FIL and TEST2.FIL, and will consider them resynchronized when five or more consecutive lines match:

```
FC TEST1.FIL TEST2.FIL /5
```

Wild cards may be used with the FC command to compare multiple files against a single file, or to compare multiple files against similarly named files. If a wild card is used in the first file name, all of the specified files are compared to the second file. If a wild card is used in the second file name, the corresponding value in the first file name is substituted for the wild card. The following commands demonstrate the wild card substitution:

```
FC *.BAT BATCHFIL.BAK
```

The above command compares all files with a BAT extension to the file named BATCHFIL.BAK.

```
FC AUTOEXEC.BAT *.BAK
```

The above command compares the AUTOEXEC.BAT file with the file AUTOEXEC.BAK.

```
FC *.BAT *.BAK
```

The above command compares all files with an extension of BAT, with all files that have a matching name and an extension of BAK.

The use of wild cards makes it very easy to compare all the files on two disks. The following command compares all files on drive A: with all files on drive B:

```
FC A:*.* B:*.*
```

The use of the MORE filter and the output redirection symbol (>) can make the FC command much easier to use. The MORE filter pauses the screen and waits for a key to be pressed whenever the screen is full, and the output redirection symbol redirects the information that would appear on the screen, to a file or printer. The following three commands demonstrate; the first uses the MORE filter to pause the display, the second sends the output to a file named FC.OUT, and the third sends the output to the printer:

```
FC TEST1.FIL TEST2.FIL | MORE

FC TEST1.FIL TEST2.FIL > FC.OUT

FC TEST1.FIL TEST2.FIL > PRN
```

If no differences are found between the two files, FC displays the message "FC: no differences encountered."

FC is an external command.

The **XCOPY** command, which is similar to the COPY command, adds the ability to copy subdirectories and the files they contain. The syntax of the XCOPY command is:

```
XCOPY source target switches
```

If no switches are specified, XCOPY copies only the files in the directory specified by **source**. If source contains only a file name specifier, with no directory, XCOPY uses the current directory as the source.

If no switches are specified, XCOPY functions similarly to the COPY command, in that it copies only files from the source directory, but not its subdirectories. However, because of its sensitivity to directory structures, XCOPY is more careful than COPY about determining whether the target specifier is a file name or a directory name. For example, consider the following COPY command:

```
COPY TEST1.FIL TEMP
```

If the current directory does not contain a subdirectory named TEMP, this command copies the file TEST1.FIL to a file named TEMP in the current directory. If there is a subdirectory named TEMP, TEST1.FIL file is copied to that directory, and the name TEST1.FIL is preserved in the copy. However, if the same command is entered with XCOPY instead of COPY, and no subdirectory named TEMP exists, the effect is different: XCOPY prompts you with the following message:

```
Does TEMP specify a file name
or directory name on the target
(F = file, D = directory)?
```

If you press F, the file is copied to a file named TEMP in the current directory. If you press D, a directory named TEMP is created as a subdirectory of the current directory, and TEST.FIL is copied into that directory. This feature is effective for any file specifier that does not end with a backslash. For example, if no there is no directory named \WP\LETTERS\NEWFILES, then the following command will cause the File or Directory prompt to appear:

```
XCOPY TEST1.FIL \WP\LETTERS\NEWFILES
```

To override this prompt, and force the creation of a new directory if the specified directory does not exist, a backslash is appended to the final directory name. The following command will copy TEST1.FIL into the \WP\LETTERS\NEWFILES directory, creating the directory if it does not exist, without a prompt:

```
XCOPY TEST1.FIL \WP\LETTERS\NEWFILES\
```

The reason this works is that the final backslash clearly defines NEWFILES as a directory, file names cannot end with a backslash. Without the backslash, there is some ambiguity as to whether the word NEWFILES represents a file name or a directory name, and the only way to resolve this ambiguity is for XCOPY to ask you what you mean.

XCOPY can be used to back up files. The /A switch is used to tell XCOPY to copy only those files that have their archive file attribute set. If the archive attribute is set, it means the file has changed since the last BACKUP was performed on it. When the BACKUP command is used, it turns off the Archive attribute; as soon as the file is modified in any way, DOS turns the archive attribute back on. Consequently, by using the /A switch with XCOPY, you can copy only those files that have changed since the last backup. XCOPY automatically turns on the Archive attribute on the target file, but it leaves the source file Archive attribute unchanged.

If you want XCOPY to turn off the Archive attribute on source files, the /M switch should be used instead of the /A switch. For example, the following two commands copy all files in the current directory that have their Archive attribute on. However, the first command leaves the source file Archive attribute on, while the second turns it off:

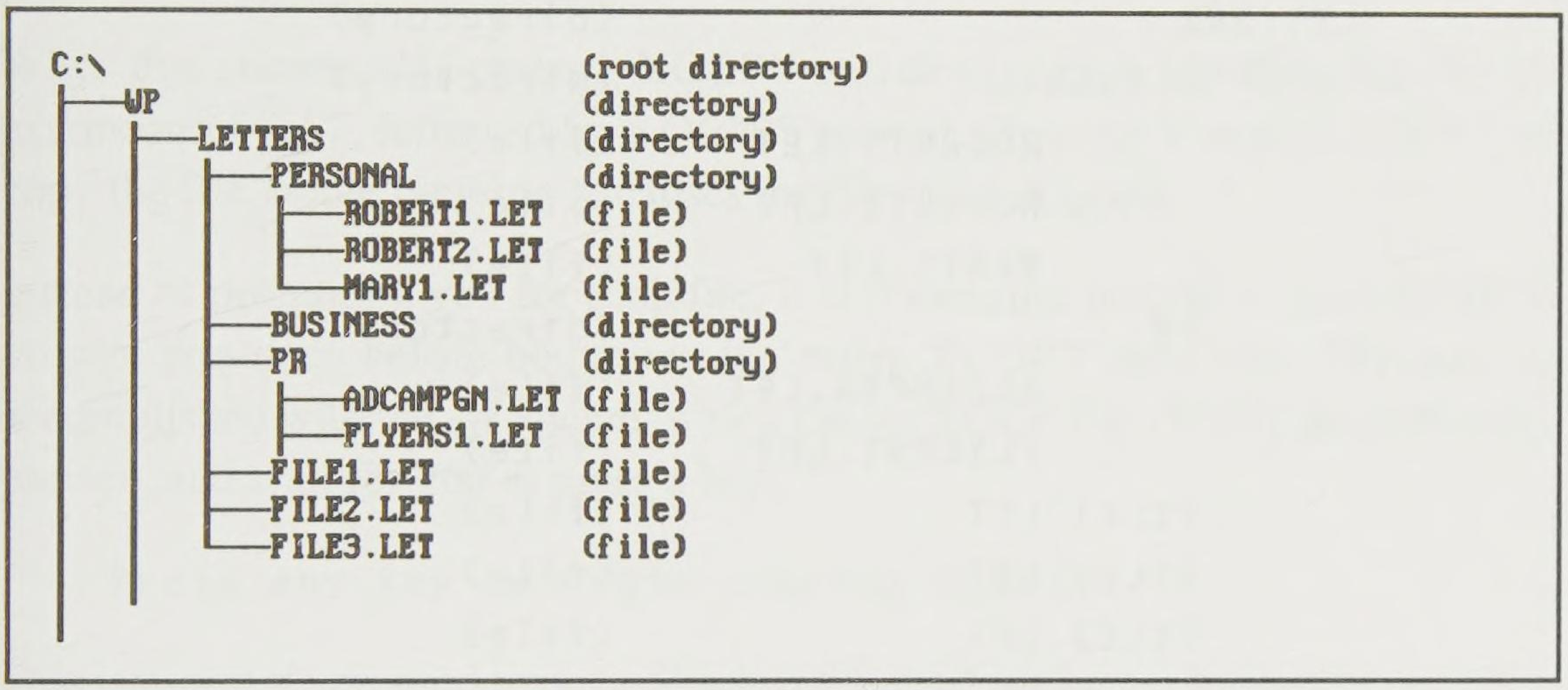
```
XCOPY *.* /A
```

```
XCOPY *.* /M
```

The /D:date switch is to copy only those files that were changed on or after the specified date. The following command copies only those files in the current directory that have an extension of LET, and that were modified on or after 8/1/91:

```
XCOPY *.LET *.BAK /D:8/1/91
```

The /S switch copies subdirectories as well as files. If a subdirectory on the source does not exist on the target, the /S switch will cause it to be created on the target. However, only subdirectories that have files in them are created. If the subdirectory on the source is empty, then it will not be created on the target. The target may be another disk or another directory on the source disk. For example, consider the partial directory structure shown in Figure 5-7.



```
C:\                          (root directory)
├──WP                        (directory)
│    ├──LETTERS              (directory)
│    │    ├──PERSONAL        (directory)
│    │    │    ├──ROBERT1.LET  (file)
│    │    │    ├──ROBERT2.LET  (file)
│    │    │    └──MARY1.LET    (file)
│    │    ├──BUSINESS        (directory)
│    │    ├──PR              (directory)
│    │    │    ├──ADCAMPGN.LET (file)
│    │    │    ├──FLYERS1.LET  (file)
│    │    ├──FILE1.LET       (file)
│    │    ├──FILE2.LET       (file)
│    │    └──FILE3.LET       (file)
│    │
│
```

Figure 5-7 — Partial File and Directory Structure.

The C: \ WP \ LETTERS directory contains three files: FILE1.LET, FILE2.LET, and FILE3.LET; and three subdirectories: PERSONAL, BUSINESS, and PR. The PERSONAL subdirectory contains three files of its own: ROBERT1.LET, ROBERT2.LET, and MARY1.LET, while the BUSINESS subdirectory is empty, and the PR subdirectory contains two files: ADCAMPGN.LET and FLYERS1.LET.

XCOPY can be used with the /S switch to duplicate the files and non-empty subdirectories of the C: \ WP \ LETTERS directory to the A: drive with the following command:

```
XCOPY C:\WP\LETTERS A:\ /S/E
```

The backslash on the target tells XCOPY to copy files from C:\WP\LETTERS into the root directory on the A: drive, and create subdirectories of C:\WP\LETTERS in the A: root directory. If the target disk was empty when this command was entered, it would end up with the structure shown in Figure 5-8:

```
A:\                               (root directory)
      LETTERS                     (directory)
            PERSONAL              (directory)
                  ROBERT1.LET     (file)
                  ROBERT2.LET     (file)
                  MARY1.LET       (file)
            PR                    (directory)
                  ADCAMPGN.LET    (file)
                  FLYERS1.LET     (file)
            FILE1.LET             (file)
            FILE2.LET             (file)
            FILE3.LET             (file)
```

Figure 5-8 — File Structure for the Target Drive.

If you wanted to copy the empty BUSINESS directory as well, the /E switch could be used in conjunction with the /S switch. /E tells XCOPY to copy all subdirectories, regardless of whether or not they are empty. The above command would be modified as follows:

```
XCOPY C:\WP\LETTERS A:\ /S/E
```

The /E switch cannot be used unless the /S switch is also specified.

The /P switch tells XCOPY to prompt you before copying each file. This is useful when you want to copy a lot of files, but they do not have enough common features in their names to use wild cards to specify them as a group. For example, you may have 20 files with a BAT extension, but only want to copy 12 of them. Rather than typing in 12 file names, the following command could by used:

```
XCOPY *.BAT *.BAK /P
```

When this command is entered, XCOPY will display each file name that has an extension of BAT, followed by a (Y/N)? prompt. Pressing Y causes XCOPY to copy the file, while pressing N causes the file to be skipped.

Instead of prompting you for each file, it is sometimes helpful to have XCOPY prompt you once before beginning the entire XCOPY operation. This can be accomplished with the /W switch, which causes XCOPY to display the following prompt, and wait for you to press a key:

```
Press any key to begin copying file(s)
```

This is useful if you need to put a disk in a floppy drive before letting the XCOPY operation continue. The /W switch is commonly used in batch files that include the XCOPY command for a floppy drive.

As with the COPY command, it is rare that XCOPY does not accurately duplicate a file. However, with very important information, or under circumstances in which a bad copy would create serious problems, you may want to use the /V switch to cause XCOPY to verify each file copied. This will slow down the performance of XCOPY, but decreases the likelihood of a bad copy.

As with the DISKCOPY command, XCOPY sets the ERRORLEVEL environment variable when it terminates. There are four possible numbers to which XCOPY may set the ERRORLEVEL variable. Their values and meanings are shown in Table 5-5.

Table 5-5 — Values of ERRORLEVEL Set by XCOPY Command.

Value	Meaning
0	XCOPY was successful; no errors occurred.
1	There were no files that matched the source specifier, so no files were copied.
2	XCOPY was terminated by the user pressing Ctrl-C or Ctrl-break
4	An initialization error occurred. This is caused by one of the following problems: insufficient memory; insufficient disk space; an invalid drive specifier; or a syntax error in the XCOPY command.
5	A disk write error occurred.

The ERRORLEVEL variable can be referenced on the IF line of a batch file to take an appropriate action if XCOPY returns a value other than 0. Batch files are discussed in Chapter 8.

For copying the entire contents of a disk, XCOPY is generally preferred over DISKCOPY for two reasons: XCOPY unfragments the files as it copies them, while DISKCOPY duplicates fragmentation; and XCOPY does not require that the disks be the same format, while DISKCOPY does. However, hidden and system files are not copied with XCOPY, while they are copied with DISKCOPY. If you want to use XCOPY to duplicate a disk that has hidden and/or system files, the ATTRIB command may be used to change these attributes before the XCOPY operation, then again to change them back after XCOPY. When XCOPY has finished copying a file, it turns on the Archive attribute in the target file, even if that attribute is not set in the source file. DISKCOPY leaves the archive attribute unchanged in the target. File attributes and the ATTRIB command are discussed in the Advanced Topics section of this chapter.

XCOPY is an external command.

Renaming Files

The **RENAME** command renames files. A drive and path specifier may used for the source file, but the target specifier must contain only a file name. The file is being renamed, not copied, so it must remain in the same directory. The RENAME command has the following syntax:

```
RENAME source target
```

Wild cards may be used with RENAME. For example, the following command uses the "many" wild card (*) to rename all files that have a BAT extension to a BAK extension:

```
RENAME *.BAT *.BAK
```

The target file name cannot already exist. RENAME will not overwrite an existing file, as COPY does. Instead, when RENAME encounters a target file name that already exists, it displays the following message:

```
Duplicate file name or file not found
```

This message also appears if the source file does not exist.

The "single character" wild card (?) may also be used with the RENAME command, but it only works if the source and target file names have the same number of characters in them. For example, suppose there were three files named TEST1.FIL, TEST2.FIL, and TEST3.FIL, in the current directory. These three files could be renamed to TEMP1.FIL, TEMP2.FIL, and TEMP3.FIL with the following command:

```
RENAME TEST?.FIL TEMP?.FIL
```

However, consider the following command:

```
RENAME TEST?.FIL TST?.FIL
```

RENAME will rename the first file, TEST1.FIL, to TSTT.FIL. It copies the first three letters from the first file name, then takes the next letter (T) to match the wild card in the target, ending up with TSTT.FIL. Now RENAME will attempt to copy the second file, TEST2.FIL, but the same thing happens. This time

TSTT.FIL already exists, so RENAME displays the message "Duplicate file name or file not found" and terminates the operation.

A similar problem arises if the target file name has more characters than the source file name. Consider this command:

```
RENAME TEST?.FIL TESTER?.FIL
```

RENAME renames the first file, TEST1.FIL, to TESTER.FIL. It replaces the first four characters of the source file with the first four characters of the target file, then matches the wild card specifier in the source file name with the next character in the target file name (an E). It then continues to use the rest of the target file name (an R), and adds on the extension, resulting in TESTER.FIL. Now, RENAME will attempt to copy the second file, TEST2.FIL, but the same thing happens. This time TESTER.FIL already exists, so again the error message is displayed and RENAME terminates. Consequently, when using the single wild card with RENAME, you must make certain the number of characters in the source and target names are identical.

DOS 5.0 does not have a command for renaming directories, but the XCOPY command can be used instead to perform a complete copy of one directory into another, with the desired name. Files in the source directory are then deleted with the DEL command, and the source directory removed with the RD command.

RENAME is an internal command.

Commands for Locating and Displaying Files

The **DIR** command displays all of the file names and subdirectories in a directory. It has a variety of switches for selecting the particular files that will be shown, and can also be used to search through subdirectories for one or more files. DIR is discussed in detail earlier in this chapter, in the section titled, "The DIR Command."

The **FIND** command locates one or more files that contain a particular text string, and for displaying the lines in the files that contain the string. There are several switches that can be used to adjust the case sensitivity of the command, to add line numbers to the displayed lines, and to have it display only lines that contain the specified string, only lines that do not contain it, or only a count of the lines that contain it. FIND is discussed in detail in Chapter 3 in the section titled, "The DOS Filter Commands — FIND, MORE, and SORT."

The **MORE**command is a filter for other commands, or as a command in its own right. When used as a command, it requires the input redirection symbol (<) to send it a file name. This displays the contents of the file on the screen, pausing after each screen until you press a key. This is the same effect achieved by the TYPE command combined with the MORE filter. TYPE is discussed next.

There are no switches for the MORE command. MORE is discussed in detail in Chapter 3 in the section titled, "The DOS Filter Commands — FIND, MORE, and SORT."

The **TYPE** command displays the contents of an ASCII (or extended ASCII) text file on the screen. It has one required parameter, and no switches. The parameter is the name of the file to display. The following command displays the contents of the AUTOEXEC.BAT file:

```
TYPE AUTOEXEC.BAT
```

ASCII text files are files that contain only the standard ASCII codes — those numbered from one to 127. If the file contains extended codes (those numbered from 128 through 255), strange characters will appear on the screen. Furthermore, if the file is produced with a word processor or other software package that provides special formatting codes, it will also display these as strange characters, and line breaks may not be shown at the appropriate places. Files created with the DOS 5.0 text editors, MS-DOS EDITOR and EDLIN, will be properly displayed with the TYPE command. The only difficulty that could arise with these files is that they may contain lines that are more than 80 characters wide. Since there are only 80 characters on the screen, the longer lines will wrap at the right edge of the screen.

Attempting to display binary files — those that have extensions of BIN, COM, EXE, LIB, or OBJ — will not damage them, but the display will not be readable.

Since most files contain more information than will fit on one screen, it is usually useful to combine the TYPE command with the MORE filter:

```
TYPE REPORT.FIL | MORE
```

This will cause the display to pause and wait for you to press a key, whenever the screen is full.

The TYPE command can also be used to send the contents of a file to the printer, using the output redirection symbol (>) and the standard device name for the printer (PRN). The following command will print the contents of the AUTOEXEC.BAT file:

```
TYPE AUTOEXEC.BAT > PRN
```

Type is an internal command.

File Protection and Recovery Commands

It is inevitable that sooner or later you will have a problem with an important file — you will accidentally erase it, and be unable to recover it with the UNDELETE command; or something will go wrong with your computer that damages the disk that contains the file; or your computer will be stolen, and the disk along with it; or some other such unforeseen event. Recognizing the inevitability of such events, the designers of DOS built a number of commands into DOS 5.0 for protecting yourself against such happenings.

Commands used to protect against accidental loss are all based on the same principle: they create a copy of the file to be protected, so that if the file is lost or damaged the copy can be used to restore it. There is a variety of ways to manage the creation of such copies, and each command takes a somewhat different approach. In addition to the commands discussed, the simple COPY and XCOPY commands discussed earlier are effective for small numbers of files, if the target disk has sufficient space to hold them.

The **BACKUP** command is useful for backing up large numbers of files and subdirectories all at once. BACKUP uses a special data compression technique to efficiently pack the files onto the target disk. BACKUP is used to back files up from one drive to another. When the target is a floppy drive and is not big enough to contain all of the source files, BACKUP will use multiple floppy disks. The simplest form of the BACKUP command is:

```
BACKUP source target
```

Where the **source** specifier may contain a drive and/or a directory specifier, as well as a file name specifier. The **target**, however, can consist only of a drive specifier. For example, the following command backs up everything in the root directory of the C: drive to floppy disks in the A: drive:

```
BACKUP C:\*.* A:
```

When this command is entered, BACKUP displays the following message:

```
Insert backup diskette 01 in drive A:

WARNING! Files in the target drive
A:\ root directory will be erased
Press any key to continue...
```

After you have placed a disk in the target drive and pressed a key, BACKUP displays the following message:

```
*** Backing up files to drive A: ***

Diskette Number: 01
```

BACKUP then proceeds to list the names of the files it is backing up, as they are added to the backup disk. If the disk fills up, BACKUP prompts you for disk 2, and continues.

BACKUP does not simply copy files from the source to the target. Instead, it combines the files, using a file compression technique that reduces the amount of space required by the files. When BACKUP runs out of space on one disk, it stops compressing the files and prompts you to put another disk in the drive. It then begins compressing again. It can interrupt the compression process even in the middle of a file. This assures that every byte of space on the target disk will be used. If you use the DIR command to examine the contents of disk 1, you will see that it contains just two files: BACKUP.001 and CONTROL.001. Disk 2 will have files named BACKUP.002 and CONTROL.002, and so on. The

BACKUP.XXX file contains the actual file contents, while CONTROL.XXX contains information used by the RESTORE command to correctly decipher the information in BACKUP.XXX, and restore it to the correct files and directories.

BACKUP continues prompting you for more disks until all of the specified files have been backed up. It is important to number the disks as you go, so that you will know what order to restore them in.

When BACKUP is used without switches, it simply backs up all of the files in the specified directory. It is often desirable to back up subdirectories, as well. This can be done with the /S switch. The following two commands demonstrates this: the first backs up all files in all directories on the C: drive, while the second backs up all files in the C:\WP\LETTERS directory, as well as all files in any subdirectories of C:\WP\LETTERS.

```
BACKUP C:\*.* A: /S
```

```
BACKUP C:\WP\LETTERS\*.* /S
```

When BACKUP backs up a file, it turns off the archive attribute for that file. If the file is modified at a future time, DOS will automatically turn the archive attribute back on. Thus, the archive attribute always indicates whether or not a file has been modified since it was last backed up. The /M switch uses the archive bit to only back up files that have changed since the last backup. With large hard disks this makes it easy to develop an efficient means of maintaining backups without wasting a lot of time repeatedly backing up files that have already been backed up, since only a handful of files out of thousands may have changed. The following command backs up all files on drive C: that have changed since the last backup:

```
BACKUP C:\*.* A: /S/M
```

The /D switch is used to back up only files that were changed on or after a specified date. The following command backs up all files in the root directory of the C: drive that were changed on or after 6/1/91:

```
BACKUP C:\*.* A: /D:6/1/91
```

In addition to specifying the date, you may also specify the time. The following command specifies that all files modified at or later than 2:00 PM be backed up:

```
BACKUP C:\*.* A: /T:14:00
```

Normally the time switch is used in conjunction with the date switch, but it may be used by itself if you like.

The BACKUP command provides the ability to create a "log" file whenever a backup is performed. The log file is a listing of each file that was backed up, the date and time at which they were backed up, and the disk number that the file was stored on. To create a log file, the /L switch is used. The following command backs up all files in the C:\WP\LETTERS directory onto the A: drive, and creates a log file:

```
BACKUP C:\WP\LETTERS\ /L
```

The log file is named BACKUP.LOG, and is placed in the source directory — in this example, in the C:\WP\LETTERS directory. You may also specify a different drive, directory, and/or name for the log file:

```
BACKUP C:\WP\LETTERS\ /L:D:\BACKLOGS\BACK10.LOG
```

This places the log file in the D:\BACKLOGS directory, and names it BACK10.LOG. The log file should not be created on a floppy drive — it should always go to a hard drive. However, after it has been created it can be copied to a floppy disk. If the log file already exists, BACKUP adds the new information to it. The log file can be very useful if, in the future, you need to remember which files were backed up.

BACKUP normally deletes any files that already exist in the root directory of the target disk. Files in subdirectories are not affected. However, sometimes it is desirable to backup files to a floppy disk without deleting files in the root directory. There are typically two reasons for doing this: in order to create a disk that contains copies of files in both backup and standard formats; and to add files to a backup file that already exists on a disk. The former is commonly done in order to provide a simple batch file that someone can use to restore the files on the disk, without having to know the RESTORE command. The latter is done when a lot of files have already been backed up, and you want to continue with the same set of disks, rather than have two different sets of backup disks.

The /A switch is used to accomplish both of these tasks — it leaves files in the root directory unchanged, and it adds files to an existing backup file. The following command backs up the files with a LET extension in the C:\WP\LETTERS directory and subdirectories, and adds them to an existing set of backup disks:

```
BACKUP C:\WP\LETTERS\*.LET /S /A
```

When this command is entered, BACKUP will display the following prompt:

```
Insert last backup diskette in drive A:

Press any key to continue...
```

The last backup diskette used in the previous sequence must be inserted in the A: drive. If a diskette that does not contain BACKUP files on it is used, BACKUP will display the message "Last backup diskette not inserted" and terminate.

Note: Since BACKUP requires that a disk have backup files on it before the /A switch can be used, the only way to save a non-backup file in the root directory is to first create a small backup file on the disk, leaving enough space for the non-backup file. Then copy the non-backup file to the root directory of the backup disk, and use the BACKUP command with the /A switch to put the rest of the files into the backup file.

If the target disk is not formatted, BACKUP automatically formats it before attempting to write information to it. However, there are different capacities, or densities, that a disk can be formatted to, and BACKUP assumes the highest possible capacity that the backup drive can handle. This is not always desirable because you may need to restore your disks on a computer that has a lower capacity drive. In this case, the /F:XXX switch may be used to specify a particular density to which the disk should be formatted. Table 5-6 shows the various disk sizes and capacities, and the possible values that may be used for XXX in the /F switch.

Table 5-6 — Formatting Values for BACKUP Command.

Disk Size	No. of Sides	Capacity	XXX Value
5-1/4"	1	160K	160
5-1/4"	1	180K	180
5-1/4"	2	320K	320
5-1/4"	2	360K	360
5-1/4"	2	1.2M	1200
3-1/2"	2	720K	720
3-1/2"	2	1.44M	1440
3-1/2"	2	2.88M	2880

For the XXX values shown in Table 5-6, you may append the letter K or the letters KB to the number. Thus, to specify that a disk be formatted to 720K, you could use any of the following forms of the /F switch:

```
/F:720
/F:720K
/F:720KB
```

If /F is used without a colon and an XXX value, BACKUP uses the default value for the target drive. For BACKUP to format a disk, the DOS FORMAT.COM program must either be in the current directory, or in a directory specified in the PATH.

GOOD IDEA

It is important to develop a plan for periodic backups. This should include occasional complete backups of your entire hard disk, as well as more frequent partial backups of files that have changed. In general, the following scheme works well; though the exact time periods may need to be adjusted to reflect the frequency with which you use your computer:

Once a month, perform a complete backup of the entire hard disk. Store this in a safe place, preferably in a different building from the computer, in case of a fire or other catastrophic disaster.

Once a week, perform an incremental backup, backing up only those files that have changed since the last backup.

Keep two generations of each set of disks: at the end of the first month, use a new set of disks for the complete backup. At the end of the second month, re-use the first month's disks, and so on. Follow a similar pattern for the weekly backups.

There are a few points of caution to keep in mind with the BACKUP command: it cannot back up the system files, IO.SYS, MSDOS.SYS, and COMMAND.COM. However, these files can be transferred to a floppy disk with the SYS command. The BACKUP command provided with DOS 5.0 cannot be used with a RESTORE command from DOS 3.2 or earlier. The format of the backup files changed with DOS 3.3. It has remained consistent since then, so a RESTORE command from DOS 3.3 or later will be consistent with DOS 5.0. However, the DOS 5.0 RESTORE command can be used with backup disks made with the BACKUP command from any version of DOS.

Since BACKUP encodes files, there is an increased likelihood of a backed up file having problems — each time an additional step is added to the copying process, the likelihood of errors increases. Consequently, for small numbers of files it is best to maintain your backups as normal copies of files. The COPY or XCOPY commands can be used for this purpose, but the REPLACE command, discussed later in this chapter, is usually better. When using the BACKUP command, it is wise to turn on the VERIFY switch, discussed below, to add an extra measure of protection against bad backup disks. When using COPY or XCOPY to back up files, you can use the /V switch.

Note: If you have used the ASSIGN, JOIN, or SUBST commands to redirect a drive, you should not use the BACKUP command with that drive. It should be restored to its original configuration before BACKUP is used with it. Otherwise, the RESTORE command may not be able to restore the files.

BACKUP leaves a return code in the DOS environment variable ERRORLEVEL. Table 5-7 shows the return code values and their meanings for the BACKUP command.

Table 5-7 — Values of ERRORLEVEL Set by XCOPY Command.

Value	Meaning
0	BACKUP was successful; no errors occurred.
1	There were no files that matched the source specifier, so no files were backed up.
2	File-sharing conflicts on a network prevented some files from being backed up.
3	BACKUP was terminated by the user pressing Ctrl-C or Ctrl-break
4	BACKUP terminated because of an error.

The BACKUP command is an external command.

The **RESTORE** command restores files from disks created with the BACKUP command. The syntax is therefore the reverse of that for BACKUP: the source specifier can only be a drive letter, while the target may contain drive, path, and file name specifiers:

```
RESTORE source target
```

Where the **source** is the drive on which the backup files are stored, and **target** is the drive to which those files should be restored. It can be a little confusing at first to understand why the target should have drive and file name specifiers, since the files exist on the source drive. Recall that when BACKUP backs up files, all of the files are compressed into one single backup file in the root directory of the backup disk. Consequently, there is no directory name to specify for the RESTORE source disk. In fact, there isn't even a file name, since the file names are all encoded along with the directory names. Furthermore, if a file was in a directory named \WP\LETTERS\BUSINESS on the drive that was backed up, then RESTORE can only place it into a directory named \WP\LETTERS\BUSINESS on the target drive. For these reasons, the directory and file name specifiers are always applied to the target drive in the RESTORE command.

You may use file names and wild cards to restore only selected files from a backup disk. For example, the following command restores the file ROBERT1.LET in the \ WP \ LETTERS \ PERSONAL directory:

```
RESTORE A: C:\WP\LETTERS\PERSONAL\ROBERT1.LET
```

By default, the RESTORE command restores files only to the specified directory. The following command restores all files that were backed up from the root directory of drive C:, but does not restore subdirectories:

```
RESTORE A: C:\*.*
```

To restore subdirectories of the specified directory, use the /S switch. The following command restores all of the files that were backed up from any directory in drive C:

```
RESTORE A: C:\*.* /S
```

RESTORE can be used to list the files that are backed up on a backup disk. This is particularly useful if no log file exists for the backup disks. To display a list of the files on a backup disk, enter the following command:

```
RESTORE A: C: /D
```

The /D switch disables the actual restoration of files, so that the file names are simply listed out on the screen. Nonetheless, the command still requires that a target drive be specified. The listing of files can also be routed to a file or printer, instead of the screen. The following command prints the list of files:

```
RESTORE A: C: /D > PRN
```

WARNING

If RESTORE encounters a file on the target disk that has the same name as one being restored, the file on the target disk is replaced with the one from the backup disk. This can create problems — the one on the target disk may in fact be the preferred version of the file. Consequently, RESTORE includes switches that let you set a level of protection against accidental replacements.

If a file exists on the target disk that has the same name as one being restored, the /P switch will cause RESTORE to check the read-only and archive attribute settings for the file on the target disk. If the file's read-only attribute is on (which means the file cannot be modified by most commands or programs), or if the archive attribute is on (indicating that the file has been changed since its last backup), then this switch causes RESTORE to pause and prompt you for confirmation before replacing these files.

You can avoid overwriting any files at all by using the /N switch, which tells RESTORE to restore only those files that do not exist on the target disk.

The **/B:DATE** switch is similar to the /D:DATE switch on the BACKUP command. However, where /D:DATE tells BACKUP to back up files modified on or AFTER the specified date, /B:DATE tells RESTORE to restore only those files that were modified on or BEFORE the specified date. The date on the /B switch applies to the dates of files on the target disk, not the backup disk.

The **/A:DATE** switch tells RESTORE to restore only those files whose date on the target disk was modified on or after the specified date.

The **/E:TIME** switch tells RESTORE to restore only those files whose time of last modification was at or earlier than the specified time.

The **/L:TIME** switch tells RESTORE to restore only those files whose of last modification was at or later than the specified time. As with the date specifiers, the time refers to the time stamp of the file on the target disk, not the backup disk.

The **/M** switch restores only those files modified since the last backup (those whose Archive attribute is on).

The RESTORE command leaves a return code in the DOS environment variable, ERRORLEVEL. Table 5-8 shows the return code values and their meanings for the RESTORE command.

Table 5-8 — Values of ERRORLEVEL Set by RESTORE Command.

Value	Meaning
0	RESTORE was successful; no errors occurred.
1	There were no files that matched the target specifier, so no files were restored.
3	RESTORE was terminated by the user pressing Ctrl-C or Ctrl-break
4	RESTORE terminated because of an error.

RESTORE is an external command.

The **MIRROR** command provides an automatic measure of protection for recovery of accidentally deleted files, or an accidentally formatted disk that has valuable files on it. To understand how MIRROR works, it is necessary to understand a little about how directories keep track of files. This material is covered in detail in Chapter 6, so the discussion here will be brief.

Keeping track of directories actually involves two types of structures created by DOS. The first is a table known as the FAT or **F**ile **A**llocation **T**able. The second is the directory structure. As mentioned earlier in this chapter, when a file is stored on a disk, it may not be stored in consecutive clusters, but instead could be stored on clusters that are scattered all around the disk. The problem for DOS, then, is how to keep track of which clusters belong to which files. This information is stored in the FAT.

The FAT contains information about every cluster on the disk. Among other things, each cluster entry contains a number that indicates the next cluster in the sequence for a particular file. For example, if a file occupies clusters 9, 12, and 27, then the FAT entry for cluster 9 would contain the number 12 as the next cluster in the sequence, and the entry for cluster 12 would contain the number 27. The last cluster for the file, 27 in this example, has a special symbol that says "This is where the file ends." Thus, by starting at the first cluster for a file and following the sequence of "next cluster" numbers, it is possible to determine all of the clusters assigned to a file, and the sequence in which those clusters should be read. The only problem that remains is knowing which cluster marks the beginning of a particular file. This information is recorded in the directory. For example, a file named ROBERT1.LET, in the C: \ WP \ LETTERS directory, was stored on clusters 9, 12, and 27. The C: \ WP \ LETTERS directory stores the name of the file, information about the file's attribute settings, and the starting cluster for the file. When the file is accessed, DOS gets the starting cluster from the directory, then goes to the FAT to locate the rest of the clusters.

When a file is deleted, the information in the file is not really removed from the disk. This would take a long time, particularly for large files, and it would make file recovery impossible. Instead, two things happen when a file is deleted: the first letter of the file name in the directory is changed to ASCII code 229 (E5h) which is the Greek Sigma character. This tells DOS that that space in the directory can be used for a new file, but until it actually is used, all of the information about the old file is preserved (except for the first letter of the file name). The second thing that happens when a file is deleted is that the FAT entries for that file — the cluster map for the file — are changed to indicate that the cluster(s) are available for use. This eliminates the information that linked the clusters together for the file that was deleted. Now, when DOS wants to create a new file on the disk, it sees those clusters as available for storing the new file. However, until new information is actually written into those clusters, the old information is still there.

File recovery, then, may be fairly simple. If the file is small enough to fit in a single cluster, and neither the directory entry nor the FAT entries that the file used have been overwritten, all that is required is to change the first letter of the file name back to its original value, and change the cluster marker in the FAT back to the value that indicated the cluster was in use. Since the starting cluster number is stored in the directory, there is no problem locating it.

If the file occupied several clusters, file recovery is a little more difficult, because only the starting cluster is recorded in the directory, and the FAT pointer to subsequent clusters has been lost. If the file was not too badly fragmented — if it was stored on consecutive or nearly consecutive clusters — then it is often possible to determine which clusters belonged to it by examining the pattern of used and unused clusters in the immediate vicinity. If the file was highly fragmented, however, or if the directory entry and/or the FAT entries for the file have been overwritten, then recovery becomes more difficult.

The UNDELETE command can do a pretty good job of recovering simple file problems, since the information is still mostly available, and what's missing can be easily deduced. But as the task becomes more complex, UNDELETE by itself becomes less and less effective. This is where the MIRROR command comes in.

MIRROR is a program that can be used either as a stand-alone program, or as a TSR — a program that is stored in memory and works behind the scenes, while you go about using the computer for your work. When used as a stand-alone program, it creates two special files (named MIRROR.FIL and MIRORSAV.FIL). MIRROR.FIL is a read-only file, which means it cannot be modified or deleted unless you use the ATTRIB command to change its read-only attribute. MIRORSAV.FIL is a system file, a read-only file, and a hidden file. All of these attributes must be changed before you can delete the file. The hidden and system attributes make the file invisible to the DIR command, unless you use the /A switch.

MIRROR.FIL contains a copy of the FAT and root directory, and can be used to recover from an accidental formatting of the disk. The format recovery aspects of the MIRROR program are discussed in Chapter 6.

When MIRROR is entered by itself, with no parameters or switches, it creates the MIRROR.FIL file in the root directory of the current drive, and stores the current FAT and directory information in it. To apply MIRROR to a drive other than the current one, append the drive specifier to the command. The following two commands demonstrate this: the first one creates a mirror file for the current drive which is therefore stored in the root directory of the current drive. The second one creates a mirror file for the A: drive, which is stored in the root directory of the disk in the A: drive:

```
MIRROR
MIRROR A:
```

If the MIRROR command is used on a drive that already has a mirror file created, the old file is renamed to MIRROR.BAK. This provides a one-generation backup of mirror files, in case you need to go back and recover a file that was deleted after the first mirror file was created, but before the second one was created. There is one drawback to this: it takes up additional disk space, since there are now two mirror files instead of one. The amount of disk space depends on the size of the disk, and on the use of the /T switch. The /1 (the number one, not the letter L), switch can be used to turn off this backup feature. When /1 is used, the old mirror file is deleted instead of being renamed.

To use the MIRROR program for recovering deleted files, the /T switch is used. This loads MIRROR as a TSR, which constantly tracks file deletions. When you load MIRROR with the /T switch, you must specify which drive or drives you want MIRROR to track. This is done by following the T with the drive letter, but no colon. If you want MIRROR to track more than one drive, the /T switch must be used for each drive. The following command loads the MIRROR program into memory, and tells it to track file deletions on the C: drive and the A: drive:

```
MIRROR /TC /TA
```

It does not matter which drive you specify first. However, the drive that is current does make a difference. When this command is entered, the files MIRORSAV.FIL and MIRROR.FIL are created on the current drive. These files are used for recovery from accidental format, so even though you specified two drives for deletion tracking, format protection is only provided for the current drive. Since these two files are used only for format protection, another mechanism is employed for protection against file deletion. When MIRROR is loaded into memory with the /T switch, it immediately starts monitoring the status of all files on the specified drive(s). As soon as it detects a file being deleted, it creates a file named PCTRACKR.DEL in the root directory of the drive that holds the file. All of the information needed to recover the file — it's directory and FAT entries — are saved in PCTRACKR.DEL. PCTRACKR.DEL has its System attribute turned on, so it cannot be seen with the DIR command unless the /A switch is used, and it cannot be deleted unless the ATTRIB command is used to turn off the System attribute.

The size of PCTRACKR.DEL depends on the capacity of the drive on which it is created, and it determines the maximum number of deleted files that can be tracked. The possible disk capacities, the corresponding default sizes of PCTRACKR.DEL, and the number of deleted files that can be tracked in that size file, are shown in Table 5-9.

Table 5-9 — Disk Sizes and Corresponding Default PCTRACKR.DEL Sizes.

Disk Capacity	Default Size of PCTRACKR.DEL	Number of Files Tracked
360K	5K	25
720K	9K	50
1.2MB	14K	75
1.44MB	15K	75
20MB	18K	101
32MB	36K	202
Over 32MB	55K	303

You can adjust the size of PCTRACKR.DEL up or down by adding a hyphen (–) after the drive letter, then typing the number of deletions you want to track. The number of deletions tracked may be anywhere from one to 999. The following command loads MIRROR into memory, and sets up deletion tracking to track up to 100 deletions on the A: drive and up to 500 on the C: drive:

```
MIRROR /TA-100 /TC-500
```

If the PCTRACKR.DEL file is full, deleting another file causes the information for the first file that was deleted to be replaced with the information for the one being deleted. The next file to be deleted replaces the second file that was deleted, and so on.

The /PARTN switch is used with MIRROR to save information about how a hard disk is partitioned. This switch is discussed in detail in Chapter 6.

Since MIRROR is a TSR, it uses up memory. It can also slow down the performance of your disk drive, particularly if you are deleting a lot of files, since it must write information to the PCTRACKR.DEL file for every file that is deleted. For both of these reasons, it is sometimes desirable to remove it from memory. This can be done with the /U switch:

```
MIRROR /U
```

If you need to conserve memory as much as possible, the MIRROR program can be loaded into a UMB, using the LOADHIGH command. UMBs and LOADHIGH are discussed in Chapter 4.

It is important to recover a file as soon as possible after it has been deleted. Anything you do that causes information to be written to the disk can overwrite clusters that contain some or all of the file that you want to recover. Some commands and programs create temporary files on the disk that are unknown to you. For example, the MORE command sometimes creates a temporary file to store information that will be displayed on the screen. It is often difficult to tell whether or not a program will create such temporary files, so after discovering that a file needs to be undeleted, no programs or commands other than UNDELETE should be used.

Because file recovery becomes more complex, and less likely to be successful, on highly fragmented disks, it is worthwhile to use a disk optimization utility periodically. These are programs that rearrange the files on a disk, so that all of the clusters for each file are consecutive. This not only makes file recovery simpler, it also reduces the amount of time the computer spends retrieving files, and reduces wear and tear on the disk drive. Beware, however, that once you have optimized your disk, it may be impossible to recover files that were deleted before the optimization process. There are several good disk optimization programs that can be purchased from software dealers.

MIRROR should not be used with drives affected by a SUBST or JOIN command, and if you want to use the ASSIGN command, it must be done before loading MIRROR.

MIRROR is an external command.

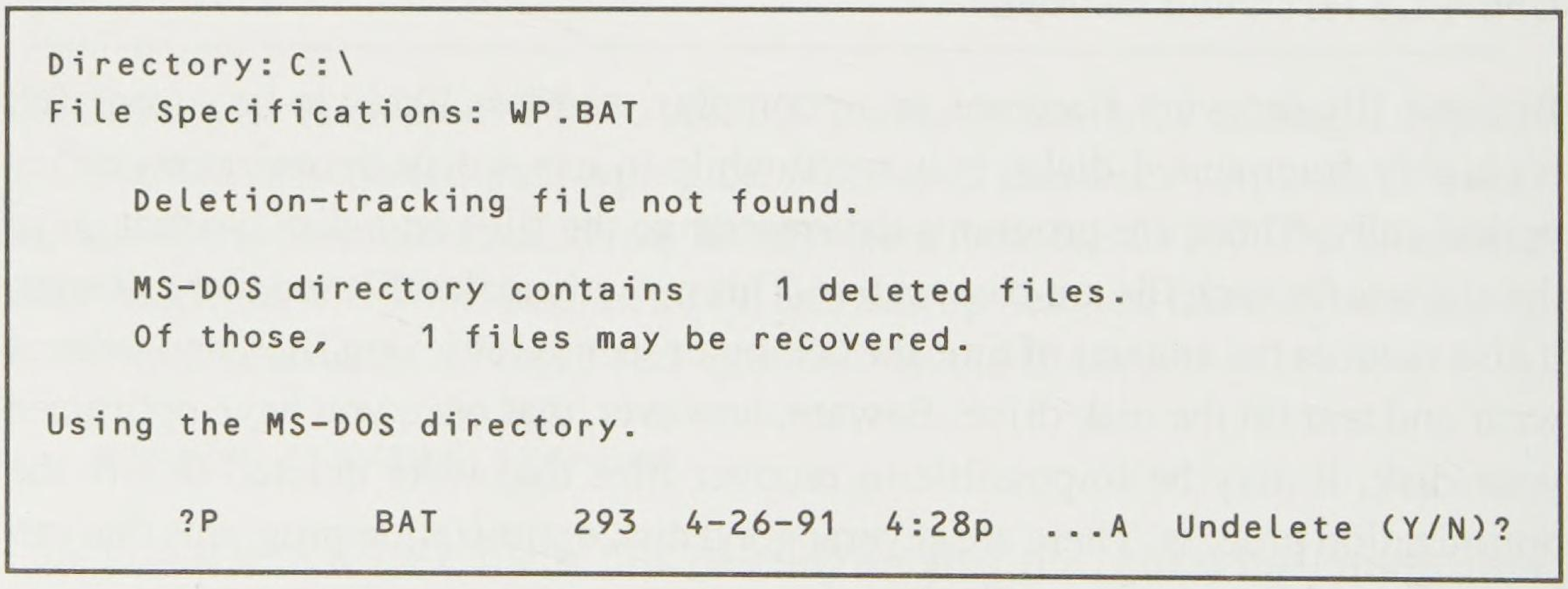
The **UNDELETE** command recovers files that have been deleted with the DELETE command or by a program such as a word processor. If you are unfamiliar with the MIRROR command, and with the mechanics of how files are deleted, you should review the preceding material on the MIRROR command. The discussion that follows assumes you are familiar with that material.

If no MIRROR files are present, UNDELETE uses the information in the current directory to attempt to identify and recover deleted files. If MIRROR files are present, UNDELETE will use them. UNDELETE is more likely to be successful if MIRROR files are present.

To recover a single file, enter the UNDELETE command, with the file name passed as a parameter. A drive and/or directory specifier may precede the file name. The following command tells UNDELETE to recover a file named WP.BAT in the root directory of the current drive:

```
UNDELETE \WP.BAT
```

What happens after this command is entered depends on whether or not MIRROR files exist. If they don't, UNDELETE displays the screen shown in Figure 5-9.



```
Directory: C:\
File Specifications: WP.BAT

    Deletion-tracking file not found.

    MS-DOS directory contains    1 deleted files.
    Of those,    1 files may be recovered.

Using the MS-DOS directory.

      ?P       BAT       293  4-26-91  4:28p  ...A  Undelete (Y/N)?
```

Figure 5-9 — UNDELETE Screen When No Deletion Tracking File is Present.

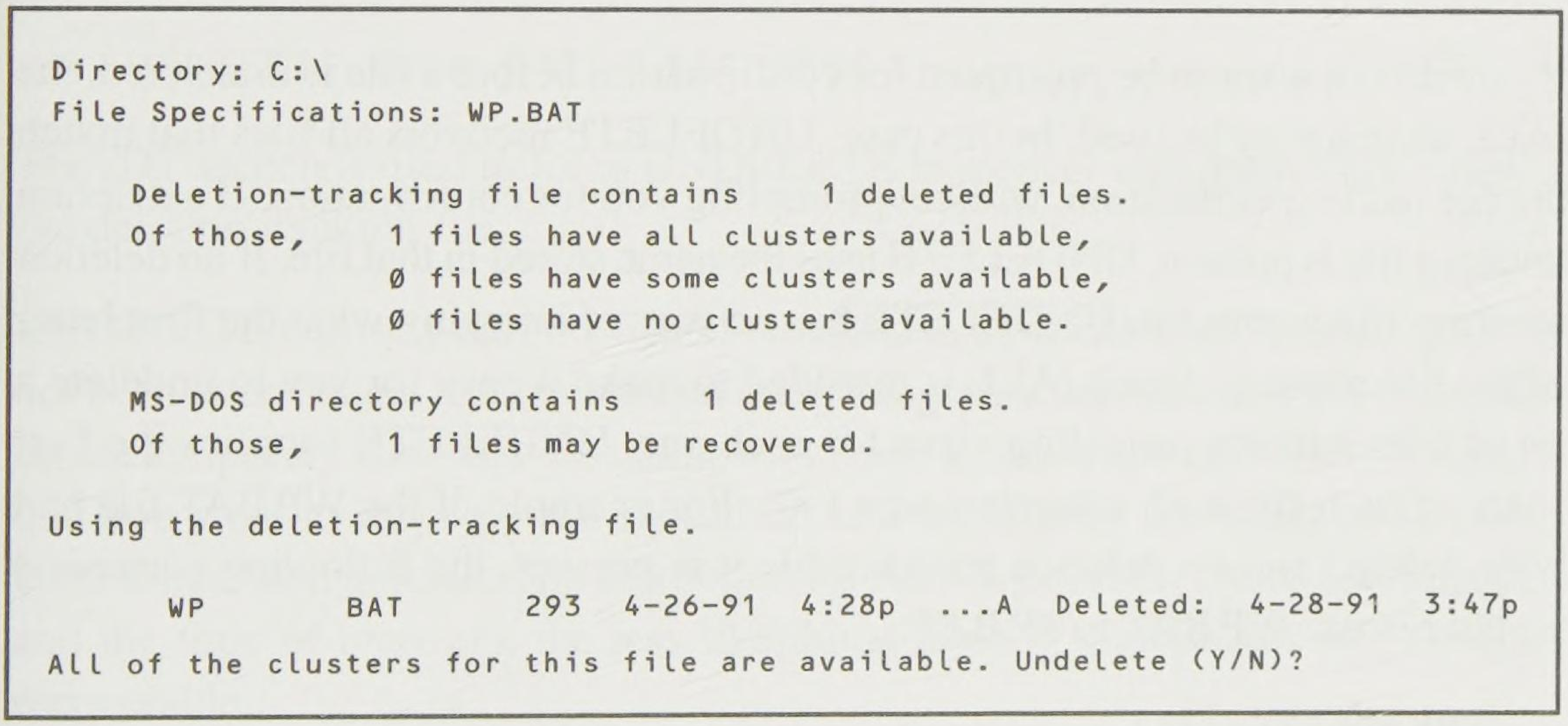
The information in Figure 5-9 indicates that no deletion tracking file is present and therefore UNDELETE is using information stored in the root directory. Recall that DOS deletes a file by changing the first letter of its name to a special code.

Since no deletion tracking file is present, UNDELETE has no way of knowing what the first letter of the deleted file is, so it shows the file name, with a question mark for the first letter. The directory information, including attribute settings (only the Archive attribute is on), is shown. You are prompted to enter a Y or N to indicate whether you want to undelete this file. If you enter Y, UNDELETE displays the following prompt:

```
Please type the first character for ?P      .BAT:
```

Whatever letter you type is used for the first letter of the file, and the file is restored. If a file of that name already exists, you will be prompted to enter a different letter.

If MIRROR files are present when the UNDELETE command is entered, the screen shown in Figure 5-10 will be displayed.



```
Directory: C:\
File Specifications: WP.BAT

    Deletion-tracking file contains    1 deleted files.
    Of those,     1 files have all clusters available,
                  0 files have some clusters available,
                  0 files have no clusters available.

    MS-DOS directory contains    1 deleted files.
    Of those,     1 files may be recovered.

Using the deletion-tracking file.

      WP       BAT       293  4-26-91  4:28p  ...A  Deleted:  4-28-91  3:47p
All of the clusters for this file are available. Undelete (Y/N)?
```

Figure 5-10 — UNDELETE Screen When a Deletion Tracking File is Present.

Figure 5-10 indicates that a deletion tracking file is present, and that it is being used for the UNDELETE operation. In this case, it is not necessary to tell UNDELETE what the first letter of the file name is — it was saved in the deletion tracking file when the file was deleted. Pressing Y causes the file to be restored. If a file already exists with the name of the deleted file, UNDELETE prompts you to enter a completely new file name — not just a new first letter, as is the case when no deletion tracking file is present.

It may be that UNDELETE will be able to identify some files that were deleted, but are not entirely recoverable because another file has overwritten some or all of the clusters that were assigned to the deleted file. In this case, these files will be listed as either having some clusters available, or having no clusters available. If some clusters are available, UNDELETE will attempt to recover as much of the file as possible. If no clusters are available, UNDELETE will inform you that it is impossible to recover any of the file. There are commercial programs that provide more complete data recovery functions than are provided with the MIRROR and UNDELETE commands. These are available from software dealers, and can be a worthwhile investment. Using these programs generally requires a fairly good understanding of the technical details of disks and files.

Wild cards may be used with the file specifier to undelete all files whose names match a particular pattern. For each deleted file that matches, you are prompted as to whether or not you want to undelete it.

If you do not want to be prompted for confirmation before a file is undeleted, the /ALL switch may be used. In this case, UNDELETE recovers all files that match the file name specification, without prompting you for confirmation. If a deletion tracking file is present, UNDELETE uses the name stored in that file. If no deletion tracking file is present, UNDELETE has no way of knowing what the first letter of the file name is. Since /ALL is intended to make it easy for you to undelete a lot of files without providing input for each one, UNDELETE supplies the first letter of each file with a number sign (#). For example, if the WP.BAT file had been deleted and no deletion tracking file was present, the following command would restore WP.BAT to #P.BAT:

```
UNDELETE /ALL
```

If there is already a file named #P.BAT, UNDELETE starts down the following list of characters, using the first one that provides a unique file name:

```
# % & - 0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P Q R S
T U V W X Y Z
```

When a deletion tracking file is present, UNDELETE attempts to recover only files that are listed in that file. This is not always desirable, since you may want to recover files that were deleted before the deletion tracking file was created.

Consequently, the /DOS switch is available to force UNDELETE to use the information stored in the directory, rather than the information in the deletion tracking file. /DOS cannot be used with the /ALL switch, which always uses the deletion tracking file.

If you want to see what files are listed as having been deleted, the /LIST switch may be used. If a deletion tracking file exists, /LIST displays the names of the files that are listed in the deletion tracking file, and a count of the files marked in the directory as having been deleted. If no deletion tracking file is present, or if the /DOS switch is also used, then /LIST indicates the file names (minus their first letter) of all files that have been marked as deleted in the current directory. A drive, directory, and/or file name specifier may be included with the /L switch. The following command lists all files that have a BAT extension that have been deleted in the root directory of the C: drive, and it uses the directory information rather than the deletion tracking file to identify those files:

```
UNDELETE C:\*.BAT /LIST /DOS
```

The /DT switch is used to force UNDELETE to recover only those files listed in the deletion tracking file.

UNDELETE can only recover files if the clusters on which they were stored have not been overwritten. Remember that many programs, and some DOS commands, create temporary files that are unknown to you. Consequently, if a file is deleted that you want to recover, you should do so immediately upon discovering that the file has been deleted. The more activity there is between the time of deletion and the time of recovery, the less likelihood there is that the file will be fully recoverable.

STOP

Be particularly cautious when removing directories from a disk. Once a directory has been removed, UNDELETE cannot recover files that were deleted from that directory, even if a deletion tracking file is present. The UNFORMAT command, discussed in Chapter 6, may be able to recover subdirectories of the root directory, but UNFORMAT must be used with extreme caution, because it can irretrievably lose data on a disk.

UNDELETE is an external command.

The **RECOVER** command extracts information from damaged disks. When a disk becomes damaged, either by physical stress or exposure to a magnetic field, parts of the files stored on it can be lost. Even though some disks are manufactured as having "life-time guarantees," all disks will fail, sooner or later. When this happens, one or more of the files on the disk becomes unreadable. DOS 5.0 therefore provides the RECOVER command to extract as much information as possible from these damaged disks. RECOVER cannot completely restore the parts of files that are written on damaged parts of the disk, but it can often restore the other parts of the file, both before and after the damaged area.

The root directory of a disk occupies a specific area of the disk. This area is set up when the disk is formatted and cannot be changed, so the root directory has a finite number of entries it can contain. The actual number of entries is dependent on the size and type of disk. (Root directory capacities are discussed in Chapter 6.)

WARNING

This limitation, on the number of entries in the root directory, is significant for using the RECOVER command, because RECOVER always restores files to the root directory. Thus, if you attempt to recover more files than will fit in the root directory, you may permanently lose any files that RECOVER attempts to put into the root directory after it is full. Therefore, RECOVER must be used carefully. It is generally best to recover just one file at a time. That file can then be moved out of the root directory before recovering the next.

LoOk

RECOVER should only be used when absolutely necessary and it should never be used to undelete a file. If a program such as a word processor, spreadsheet, or database, cannot access a file it created, then the DOS COPY command should be tried. DOS can frequently copy a file that a program cannot read, and make the copy readable by the program. If this fails, then RECOVER may be your only hope. A more disastrous occurrence is the destruction of the root directory or FAT on a disk. When this happens, none of the files on the disk can be accessed, and they must all be recovered with the RECOVER command. However, if the files were lost due to an accidental FORMAT of the disk, it may be possible to recover them using the UNFORMAT command, discussed in Chapter 6. If UNFORMAT will work, it is preferable to RECOVER.

Some files are unusable after they have been recovered, because they will only work if the entire file is intact. This is generally true of binary files — those that end with BIN, COM, EXE, LIB, OBJ, or SYS extensions. In addition, many programs, such as most databases and spreadsheets, build internal linkages in their data files. These linkages maintain the relationships between various parts of the file, and are required for the program to use the file. Consequently, these programs often cannot work with a partially recovered file, since the missing part of the file will have contained linkages to other parts of the file. In this case, the program may be able to work with the first part of the file — up to the first broken link — or it may not be able to use the file at all. In this case, you are left with the choice of either attempting to view the file with a word processor, to extract what information you can manually, and re-enter it into a new data file; or throwing the file away and starting over again. The best candidates for partial recovery are plain ASCII text files, and word processing files.

There are commercial file recovery programs available from software dealers that provide more effective and complete tools for recovering files from damaged disks than the RECOVER command. These programs also include more powerful undelete tools than the UNDELETE command does. If you have an important file that must be recovered from a damaged disk, it may be worth investing in one of these packages.

There are two forms of the RECOVER command. The first is:

```
RECOVER filespec
```

Where **filespec** identifies the file or files to be recovered. A drive identifier and directory may be included, but wild cards are not permitted. When a file is recovered, the sections of the file that are readable are pieced together and stored in a new file named FILE0001.REC, in the root directory. You can then copy this file to its original name, in its original directory, and attempt to use it with the program that created it.

The second form of the RECOVER command is:

```
RECOVER drive:
```

Where **drive** refers to a drive letter. The colon is required. This form of the command is used when the FAT and/or root directory have been damaged, and you must recover all files on the disk. This is very risky, particularly with hard disks, since the root directory can only hold a limited number of files, and there may be more than that number on the disk. It is best to use the first syntax of the command, to recover individual files, unless you are absolutely certain that the number of files on the disk does not exceed the maximum number that the root directory can hold. Capacities of root directories for different disk types are discussed in Chapter 6.

What happens to bad clusters after a file has been recovered? When RECOVER restores a file, it reads the file one cluster at a time. If the cluster is good, it copies the information in the cluster to the FILExxxx.REC file that it is building. If the cluster is damaged, RECOVER marks it as unusable in the FAT. When a cluster is marked as unusable in the FAT, DOS will not attempt to use that cluster. You can see how many bad clusters there are on a disk with the CHKDSK command. A few bad clusters are not cause for alarm; some hard drives actually come with a few clusters marked bad. But if a large percentage of your disk is ending up in bad clusters, you should have your computer (or the disk) checked for a malfunction.

RECOVER does not work on drives that have been affected by the ASSIGN, JOIN, and SUBST commands, or on network drives.

RECOVER is an external command.

The **REPLACE** command replaces files on one disk or directory with files of the same name on another disk or directory. Several switches let you control how files are replaced. The most common use of REPLACE is to replace files on one disk or directory with more recent versions of the same files on another disk or directory. If there are multiple copies of a file in different directories of a disk, REPLACE can replace each of them with an updated copy in one step. REPLACE can also add files that appear on one disk or directory, to another disk or directory on which they do not appear. REPLACE is a safer way to maintain a disk with backup copies of files than the COPY command, since REPLACE can be set to only overwrite older copies of a file.

The simplest form of the REPLACE command is:

```
REPLACE source target
```

Where **source** is the source file, that will replace the **target** file. Drive letters, directory paths, and wild cards may all be used in the source and target specifiers. For example, the following command replaces a file named ROBERT1.LET in the current directory of drive A: with a file of the same name in the \WP\LETTERS\PERSONAL directory of the C: drive:

```
REPLACE C:\WP\LETTERS\PERSONAL\ROBERT1.LET A:
```

This command replaces the file on drive A: with the file on drive C: regardless of the date and time stamps on the two files. When it is finished, REPLACE displays the message "1 file(s) replaced."

If the source file does not exist on the target drive, the source file is not copied. REPLACE displays the message "No files replaced" instead. The REPLACE command can be used to add files to a drive or directory on which they do not already exist by using the /A switch. Appending /A to the above command would cause the file on drive C: to be copied to the A: drive, but only if there was no file with the same name already on the A: drive. In this case, REPLACE displays the message "1 file(s) added."

The /U switch enables you to replace the target file only if it is older than the source file. Appending /U (for Update) to the above command would cause the file on drive A: to be replaced with the file on the C: drive, if, and only if, the file on drive C: were more recent than the file on the A: drive. REPLACE determines which file is more recent by comparing the date and time stamps of the two files. /A and /U cannot both be used on the same command line.

When replacing a large number of files with one command, it is sometimes desirable to have REPLACE prompt you for confirmation before replacing each file. This is accomplished with the /P switch. The following command replaces all files with an extension of LET on drive A: with more recent versions of the correspondingly named files on the C: drive, and prompts you for confirmation before each replacement:

```
REPLACE C:*.LET A: /U /P
```

For each file that matches the source specification, REPLACE displays the file name, followed by a (Y/N)? prompt. Pressing N causes REPLACE to skip the file, while pressing Y causes it to replace the file. When REPLACE is done, it displays a count of the actual number of files replaced.

If you attempt to replace a file whose read-only attribute is set on the target drive, REPLACE displays the message "Access denied - filespec," where filespec is the target file specifier. REPLACE then terminates and does not process any more files identified by the source specifier. If you want REPLACE to replace read-only files, the /R switch can be used. This switch should be used with caution, since files are typically marked read-only specifically to prevent them from being overwritten or deleted.

Sometimes there are multiple copies of a file in various subdirectories on a disk, and you want to update all of those copies with a single copy on another disk and/or directory. This can be done with the /S switch. The following two commands demonstrate this. The first updates all files named SALES.RPT in all directories of the C: drive with a file of the same name in the A: drive. The second updates all files named ROBERT1.LET in the C:\WP\LETTERS directory, and all of its subdirectories, with a file of the same name in the A: drive.

```
REPLACE A:ROBERT.LET C:\ /S

REPLACE A:ROBERT.LET C:\WP\LETTERS /S
```

The /S switch cannot be used with the /A switch, and it does not search subdirectories on the source drive.

If you are using a floppy drive for either the source or target, or both, the /W switch can be used to cause REPLACE to wait until you press a key. The /W switch is most often used when REPLACE is used in a batch file. The following command replaces all files on the A: drive that have a BAT extension, with the corresponding files on the C: drive, and instructs REPLACE to pause until a key is pressed before starting to copy files:

```
REPLACE C:*.BAT A: /W
```

When this command is entered, REPLACE displays the message “Press any key to continue…”. When a key is pressed, REPLACE begins searching the source for the specified files.

REPLACE cannot be used to copy files whose Hidden and/or System attributes are turned on. The ATTRIB command can be used to turn these attributes off before copying them.

The REPLACE command leaves a return code in the DOS environment variable, ERRORLEVEL. Table 5-10 shows the return code values and their meanings for the REPLACE command.

Table 5-10 — Values of ERRORLEVEL Set by REPLACE Command.

Value	Meaning
0	REPLACE was successful; no errors occurred.
2	There were no files that matched the source specifier, so no files were restored.
3	REPLACE could not locate the source or destination path.
5	The user does not have access to the target files.
8	There is not enough memory available to perform a REPLACE.
11	There was a syntax error in the REPLACE command.

REPLACE is an external command.

The **VERIFY** command adds an extra measure of protection to the process of storing information on a disk. The VERIFY command is actually a switch: it turns disk write verification on and off. Verification is off when you start your computer, until a VERIFY ON command is entered. There are three forms of the VERIFY command:

```
VERIFY

VERIFY ON

VERIFY OFF
```

The first form of the command responds with a message that indicates the status of verification: "VERIFY is on" or "VERIFY is off." The second form turns it on, and the third form turns it off.

When verification is on, every time anything is written to the disk, DOS double checks to make sure it can be read. If a sector has become damaged on the disk, but has not yet been marked as bad, DOS will normally go ahead and use the sector. You may not find out it was bad until you try to retrieve information from it. With VERIFY on, DOS will recognize a bad sector and will not write to it.

Having VERIFY on significantly slows the disk performance of a computer, so in general it is best to leave it off. However, for critical processes, it can be worth turning it on. A prime example is the use of the BACKUP command. Since backup files are your only source of recovery from disasters, it is essential that they be readable. Consequently, it is a good idea to turn VERIFY on before using BACKUP.

VERIFY is an internal command.

Commands to Create, Delete, and Navigate in, Directories

Since directories are central to the management of files on a PC, DOS 5.0 provides a variety of commands for managing directories, and for moving quickly from one directory to another.

The **CHDIR** or **CD** command is a navigation command that changes the current directory. The current directory for a drive is the one to which DOS refers whenever a command or program references the drive. Thus, by default, all read and write operations to a drive go the current directory in that drive, unless another directory is explicitly specified. CHDIR can be abbreviated to just CD, which is how it is usually used. There are three forms of the CD command:

```
CD

CD NEWDIR

CD DRIVE:NEWDIR
```

The first form of the command displays the current drive and directory. For example, if C: is the current drive, and the current directory is the root, the CD command will display the following:

```
C:\
```

Since most people set up their computer to display the current drive and directory as part of the prompt, this use of the command is usually used to quickly see what the current directory is for another drive. To see the current directory for the A: drive, the following command would be used:

```
CD A:
```

The second form of the command changes the current directory for the current drive. The following commands both assume that C: is the current drive, and that the current directory is C: \ GV. The first command changes the current directory to be \ WP \ LETTERS, while the second changes the current directory to be \ GV \ DATA:

```
CD \WP\LETTERS

CD DATA
```

It is worth analyzing how the CD command interprets the new path. CD always assumes that the current directory is the starting point, so that in the second command above, it knows that DATA is a subdirectory of the current (GV) directory. In the first command, however, the new directory path starts with a backslash, which indicates the root directory. So even though CD assumes that it should start with the current directory, the initial backslash says "Jump from the current directory to the root, then continue reading the new directory path from there." So an initial backslash can be used as a quick way to start from the root, if the new path is not in the same chain of directories as the current one. In fact, a quick way to change to the root directory is:

```
CD \
```

There is one other shorthand technique, which is used for changing to a directory that is above the current directory, but is in the same branch as the current one.

This is the use of two periods to specify the immediate parent of a directory. The following commands demonstrate, assuming C: is the current drive and the current directory is \WP\LETTERS, and using the directory tree shown in Figure 5-3:

```
CD ..

CD ..\..

CD ..\BILLS
```

The first command changes the current directory to the C:\WP directory. The second command changes the current directory to the root directory — the parent of the parent of the current directory. The third command changes the current directory to the C:\WP\BILLS directory — it first moves back to the parent of the current directory, then moves down to the BILLS directory.

DOS keeps track of a current directory for every drive on the computer. Thus, if you have a computer with a C: drive and an A: drive, DOS remembers the current path for each of these drives independently. Thus, it is possible to change the directory for a drive other than the current drive, without changing the current directory on the current drive. For example, suppose that the current directory on drive A: is the root, and the current directory on drive C: is C:\WP\LETTERS. You may want to access some files in the \DATA directory on the A: drive. For referencing one or two files, it may be easy enough to simply add the path name to the command that references the file, but if you want to repeatedly access that directory, it's easier to change the current directory on A: to \DATA. The following command accomplishes this:

```
CD A:\DATA
```

This command leaves the current directory on C: unchanged, but changes the current directory on the A: drive. Now, when a command or program references the A: drive, it will automatically look in the \DATA directory, unless another directory is explicitly specified.

CD is an internal command.

The **MKDIR** or **MD** command creates a new directory. This command enables you to build the subdirectory structures used for organizing and managing files. MKDIR can be abbreviated as MD, which is how it is usually used.

The following command creates a new directory named NEW beneath the current directory:

```
MD NEW
```

As with the CD command, you may use a drive specifier and a directory path specifier to create subdirectories on disks and directories other than the current one. The same shorthand symbols as used with CD can be used with MD: backslash (\ ) for the root directory, and two periods (..) for a parent directory.

Directory names follow exactly the same rules as file names. While it is legal to include an extension on a directory name, it has become a convention to never do so. File names are discussed earlier in this chapter, in the section titled, "File Names."

MD is an internal command.

RMDIR or **RD** The RMDIR command is used to remove or delete an existing directory. The RMDIR directory can be abbreviate to RD, which is how it is usually used. The following command removes a directory named OLD from the current directory:

```
RD OLD
```

You may specify a drive and/or path to directories in locations other than the current directory of the current drive.

A directory must be empty before it can be removed. It may not contain any files or subdirectories. If you attempt to remove a non-empty directory, RD will respond with the message "Invalid path, not directory, or directory not empty." If this message appears, and the name you specified is indeed a directory, then you should make sure you have deleted all files from the directory. If there are files you want to keep, they can be copied to another directory before deleting them. If there are subdirectories, they must be emptied of all files and subdirectories and removed as well. If you believe you have completely emptied a directory, yet continue to get this message, there may be files in the directory whose hidden and/or system attribute is on. Hidden and system files do not show up with a normal DIR command, so if these are the only files in the directory, it will appear to be empty. Furthermore, the DEL command cannot delete them. Using the DIR /A command will display these files, and the ATTRIB command can be used to turn off these attributes, so they can be deleted.

RMDIR is an internal command.

The **PATH** command sets the internal PATH that DOS will use to search for files. The internal PATH is a list of directories that DOS searches when you attempt to run a program, or enter an external DOS command. DOS can only search the PATH for files that have BAT, COM, or EXE extensions, and then only when it is trying to execute them — when the file names have been entered as commands, not as parameters to another command. The PATH command is discussed in detail in Chapter 3.

DOS only keeps track of a single path, regardless of how many drives your computer has, so a single path may contain references to more than one drive. Be cautious when including a floppy drive in the PATH, since the specified directory may only exist on a single disk that may not be in the drive. All directories on the PATH should be available at all times.

The current path can be viewed by entering the PATH command by itself:

```
PATH
```

To change the PATH, enter the command PATH followed by the list of directories that should constitute the new PATH. For example, the following command creates a path that contains the C:\WP\LETTERS, C:\GV, C:\, and C:\DOS directories:

```
PATH C:\WP\LETTERS;C:\GV;C:\;C:\DOS
```

Note that the directories are separated by a semi-colon, and that there are no spaces between the directories. The semi-colon is the only separator needed. The total number of characters in the PATH cannot exceed 127, so it is important to keep the PATH as concise as possible. If you find you cannot fit all of the necessary directories in 127 characters, you may use the SUBST command to assign a directory to a drive letter. The substituted drive letter may then be used in place of the complete directory name, thus shortening the path. However, beware that drives created by SUBST cannot be used with certain DOS commands, a few examples of which are BACKUP, RESTORE, and RECOVER. The APPEND command can also be used to effectively increase the length of the path, by using the /X:ON switch with APPEND. The SUBST and APPEND commands are discussed in detail in Chapter 6.

The existing path can be cleared completely, so that only the current directory is searched for executable files, by entering the following command:

```
PATH;
```

Regardless of what directories the path contains, DOS always searches the current directory before it searches any of the path directories. If there are two executable files in the same directory with the same name, but different extensions, DOS will determine which to execute according to the following precedence: COM files are executed before EXE files, which are executed before BAT files. This precedence can be overridden by including the extension in the command. For example, if the current directory contained three executable files named TEST.COM, TEST.EXE, and TEST.BAT, to execute the TEST.BAT file, you would enter the command `TEST.BAT` on the command line.

Since creating a new path destroys the old one, you must make sure any directories in the old path that you still want DOS to search, are included in the new one. If you have a long path, and just need to add a single directory to it, it is a nuisance to have to retype the entire path. It is not possible to directly append a new directory to the existing path, but the following batch file allows you to do this:

```
SET PATH1=%PATH%

PATH=%PATH%;%1
```

This batch file uses the DOS environment variable named PATH, which contains the current path, and the batch file parameter %1, which contains the first parameter passed on the command line when the batch file is run, to reset the

PATH with the command line parameter appended to it. If this file were named NEWPATH.BAT, then the following command would add a directory named \DB to the path:

```
NEWPATH \DB
```

The old path is saved in an environment variable named PATH1, in case you need to restore it. Batch files are discussed in detail in Chapter 8.

Another consideration in creating a path is that the longer the path is, the longer it will take DOS to find files that are not in the current directory. DOS searches the path directories in the order they appear, so you can speed up the performance of your computer by placing the directories with frequently used programs early in the path.

PATH is an internal command.

Interrupting DOS Commands

DOS commands may be interrupted by holding down Ctrl and pressing either C or break. Occasionally, Ctrl-break will work where Ctrl-C won't, but the reverse is never true, so Ctrl-break is the best. This technique should not be used too freely, since interrupting a command can leave behind partial files, which you might later think are usable. For example, interrupting the BACKUP and COPY command can result in incomplete file copying, so that only part of file is backed up or copied. However, it is extremely useful to be able to interrupt a DEL *.* command that was entered by accident.

Advanced Topics

The information covered so far in this chapter is sufficient for the file and directory management requirements of most PC users. However, understanding in more detail how directories are actually configured, and what information they keep about each file, can enhance your use of the commands discussed earlier. The ATTRIB command is also presented in detail later in this chapter.

How Files are Stored in Directories

For each file in a directory, the directory reserves 30 bytes of space, in which it stores the following eight items, in the order shown:

1. The file name, occupies the first eight bytes. Thus, the file name can be up to eight characters long. When a file name has fewer than eight characters, its name is left-justified, with the rest of the eight bytes being filled with blanks.

2. The file extension, occupies the next three bytes. Thus, the extension can be up to three characters long. When a file extension has fewer than three characters it is left-justified, with the rest of the three bytes being filled with blanks. The period that separates the file name from the file extension is not stored. DOS uses the period when file names are entered to tell where the name ends and the extension begins.

3. The file attributes occupy the next byte. The attributes will be discussed later.

4. The next 10 bytes are reserved. They are not currently in use, but the developers of DOS recognize that additional space will be needed in the future, so they've reserved it now. That way, when more information is added to the directories in the future, directories created under today's version of DOS won't have to be reconstructed.

5. The next two bytes are reserved for the time stamp of the file. The time is recorded using a code that DOS interprets when it displays a directory listing.

6. The next two bytes are reserved for the date stamp of the file. The date is recorded using a code that DOS interprets when it displays a directory listing.
7. The next two bytes are reserved for the beginning FAT entry. This number indicates the cluster that is assigned to the first part of the file. Given the first cluster, the FAT can be used to build the chain of clusters that constitute the rest of the file.
8. The last four bytes are reserved for the file size. This allows files to grow to 4,294,967,295 bytes, which is one less than four gigabytes.

File attributes are special markers that DOS uses to determine how a file should be interpreted, and what access should be granted to it. An attribute can either be on or off. An attribute that is on is said to be "set." If the attribute is on, the file is considered to have that attribute; otherwise it does not. Every file can have any of the following six attributes:

1. **Read-only**: When this attribute is set, the file cannot be deleted or modified. If an attempt is made to modify it, the message "access denied" is displayed. Files that are marked read-only are generally files that are crucial for the performance of a program, and that should never be modified.
2. **Hidden**: When the hidden attribute is set, the file is not displayed in normal DIR command listings, or when software programs display a directory listing. (Some software programs are designed to let you display directory listings that include hidden files, but most do not). Hidden files are generally files that are so vital to a program's operation that it is considered best to not even let people know they are there. Hidden files are also sometimes used in copy-protection schemes — techniques that make it difficult to duplicate a file, thus preventing software piracy.

   When a file is hidden, it is treated as a read-only file — if you know it's there, you can read it. Most programs will actually load and display a file that is marked as hidden, but if you attempt to save the file to the disk, an "Access denied" message will appear. It is interesting to set a file's hidden attribute to on, then attempt to see it with a DIR command — it doesn't show up. But when you enter a TYPE command, its contents are displayed on the screen.

3. **System**: The system attribute indicates that a file is used as part of the operating system. Thus, when you install DOS 5.0 on a disk, there are two files placed on the disk with the system attribute set: IO.SYS and MSDOS.SYS. If you use the SYS command to create a boot disk, these files are transferred to that disk. A file with the system attribute set is treated as though the hidden attribute is set. It does not show up in directory listings, and it cannot be modified. However, like a hidden file, its contents can still be viewed, if you know it's there. Some software programs create files and turn on their system attributes, even though they are not a part of DOS. These programs are generally utilities that provide deletion and formatting protection that goes beyond the protection provided by DOS.

4. **Volume Label**: The volume label attribute is used to indicate that a file should not be treated as a normal file, but instead contains the volume label for the disk. When you enter a DIR command for a disk, say the C: drive, you will either see the message "Volume in drive C has no label," or "Volume in drive C is xxx," where **xxx** represents a volume label. A volume label may be up to 11 characters long. The 11 characters may seem like a strange number because numeric limits in computers are usually in powers of two, like 8, 16, 32, or 64. However, note that 11 is the total number of characters in a file name — eight for the name, and three for the extension. When a file has the volume label attribute set, DOS combines the name and extension to generate the volume label. There should only be one such file entry per disk, and it should appear only in the root directory. In this special case, the directory entry does not apply to a true file: there is no pointer to the first cluster in the FAT, and no space is reserved on the disk for the contents of this file. Unlike read-only, hidden, and system files, the volume label file cannot be viewed in any way: the file-handling commands will treat it as though it were not there. The only way to see any part of this file is to use the DIR and LABEL commands to display and change the label itself.

5. **Subdirectory**: The subdirectory attribute tells DOS that the file is actually a subdirectory. Subdirectories are really just files, which is why there is no limit to how many entries they can hold. But unlike most files, DOS requires their contents to have a very precise format, described earlier in this chapter in the section titled, "Directories."
6. **Archive**: The final attribute is the archive attribute. This attribute keeps track of which files have been modified since their last backup. When a file is backed up, either using the BACKUP command, the XCOPY command with the /M switch, or a commercial backup utility, the archive attribute is turned off, indicating that the file has been backed up. As soon as the file is modified in any way, DOS turns the archive attribute back on, indicating that the file has changed.

You may be wondering how DOS manages to store six attributes, given that only a single byte is reserved for attribute settings. In general, a byte is considered the smallest unit of information that can be stored in a computer. But remember that a byte is actually eight bits, and that each bit can be either on or off. Each attribute setting is a single binary value, so it is possible to store up to eight attributes in this one byte. This leaves two unused attributes, and in fact there are two attributes reserved for future use.

While it is true that the computer must access an entire byte at a time, by using special binary operators built into the processor, it is possible to figure out how each bit in a byte is set. Thus, DOS can test each attribute bit individually, to determine the attribute settings for each entry in a directory.

Although each attribute has a particular meaning, DOS exercises no control over who can turn the archive, hidden, read-only, and system attributes on and off, or which files can have which attributes set. Thus, you could create a file with your word processor, then mark it as a system file. The only attributes DOS maintains control over are the label and subdirectory attributes. This is wise, because if either of these attributes is set incorrectly you could damage the disk, resulting in the loss of important files.

The ATTRIB command can be used to view those attributes that you are allowed to adjust. The syntax is:

```
ATTRIB filename
```

For example, from the root directory of a bootable disk, you could enter the command ATTRIB IO.SYS. The following response will appear:

```
SH C:\IO.SYS
```

This indicates that the file is a **S**ystem file, and that it is **H**idden. The attributes that may appear to the left of a file name are: R for read-only, H for hidden, S for system, and A for archive. The volume label and subdirectory attributes do not appear, because you are not allowed to manipulate them.

The syntax for changing a file's attributes is as follows:

```
ATTRIB +/-x filename
```

Where the **x** stands for A, H, R, or S (for **A**rchive, **H**idden, **R**ead-only, or **S**ystem). The plus sign turns on the attribute, and the minus sign turns it off. Drive letters, directory paths, and wild cards may be used as part of the file name. You may also include all of the files in subdirectories of the specified directory that match the file specifier by using the /S switch. The following command turns off the archive attribute in all files that have an extension of BAK in the C: \ WP \ LETTERS directory, and in all of its subdirectories:

```
ATTRIB -A C:\WP\LETTERS\*.BAK /S
```

The first byte of a file name has special meaning to DOS. When a file is deleted, this byte is changed from the first letter of the file name to an ASCII code 229 (E5h), which happens to be the Greek Sigma character. So if the first byte contains a legal file name character, DOS knows there's a file represented by this item. If it contains an ASCII 229, DOS knows the file has been deleted. There are two other possible values that DOS may store in the first character of a file name: a period, which indicates that DOS is using that entry for some internal data it needs to maintain and manage the directory structure; or a zero, which indicates that the entry has never been used.

When a directory is created, it is allocated one cluster of disk space, just like any other file. However, unlike other files, this cluster is then initialized with blank directory entries. All of the entries in the directory are initialized to have a zero in the first byte of the file name. When files are added to a directory, DOS places them in the directory sequentially. Consequently, DOS can always tell when it has read the entire contents of a directory: it reads until it comes to the first entry for which the first byte of the file name is zero. This is the first entry that has not been used, and is therefore at the end of the directory. As the directory grows, additional clusters are allocated and initialized as needed to store more entries.

The DOS 5.0 Shell MOVE utility transfers a file from one directory to another, but it never actually moves the physical file, as long as the file remains on the same disk. Now that it is clear how directories store file information, it is easy to understand how this can be accomplished: all that MOVE does is copy the directory entry from the source directory to the target, then delete it from the source. Some software packages also include MOVE utilities that work the same way.

Understanding the Root Directory

For most purposes, the root directory can be considered the same as any another directory. But it is unlike other directories in three important ways: it is created during the formatting process; it is placed in a special region of the disk immediately following the File Allocation Table; and it has a fixed size — it cannot grow as it fills up — it can only hold a fixed number of entries. Most hard disks have a root directory large enough to hold 512 files, including subdirectories. 360K floppies have root directories that will hold 112 entries, including subdirectories. The rest of the directory structure grows out of the root directory.

Chapter 6

Understanding and Managing Disks

Disk drives store the information and programs that Personal Computers use. There is a growing variety of disk drives found on PCs, but all of them perform essentially the same task, and almost all of them do it in essentially the same way. This chapter discusses the different types of disk drives available, their physical and logical organization, and the DOS commands that can be used to configure and use them. This is a fairly technical subject, and for most users it isn't necessary to master it — many people who use computers have little or no understanding of this subject. On the other hand, the more completely you understand how your computer works, the better you will be able to use it to its full potential.

What Disks Are and What They Do

Disks are the primary storage media for information on Personal Computers. Disks have two characteristics that make them ideal for this purpose: they are nonvolatile, which means that the information on them does not go away when the computer is turned off; and they can be manufactured in portable forms, making it easy to transfer information from one computer to another.

The Physical Structure of Disks

Most disks today are magnetic, which means they use magnetic particles to store information. However, there are alternative technologies emerging. One of these is the optical disk. This is a disk that uses the recording principles developed for musical compact disks, also referred to as Compact Discs, or CDs, in reference to their origin. Optical disks are read and written using a laser beam, and information is stored on them as tiny distortions in a plastic medium. There are two advantages to optical disks: they can store an enormous amount of information on a small, portable disk; and they are not sensitive to magnetic fields, which can destroy the information on a magnetic disk. However, there are also two drawbacks to optical disks: they are expensive to manufacture in a way that allows them to be written to more than once; and they are still significantly slower than magnetic disks. As these limitations are overcome, optical disks may become the standard technology for information storage in Personal Computers.

For the rest of this chapter, the term "disks" is used to mean magnetic disks. Other technologies that are used to produce disks use the same logical organization as magnetic disks, since all of the disk-access logic in the computer depends on this organization. Therefore, much of the material in this chapter holds true for all types of disks, regardless of the type of media.

A computer disk is a flat, spinning plate coated with tiny magnetic particles. Reading and writing information on a disk uses the same principle that audio and video tape recording uses: the magnetic particles can be arranged in a particular order that reflects some information about the world. This is the recording process, or the "write" process in computer jargon — information is written to the disk by arranging the magnetic particles. Once this has been done, the order of the magnetic particles can be analyzed to reproduce the information. With video tapes, this is the playback process; with computer disks, it is the "read" operation — the information that was stored on the disk is read by the computer.

All of the information a computer works with is stored as a series of ones and zeros, or bits. This has been said elsewhere in this book, but bears repeating because it is the single most important fact to keep in mind when understanding how computers work. To a computer, everything is handled in terms of bits.

Disks store files. A file is simply a discrete chunk of information, much like a book. It has a beginning, a middle, and an end. There are various ways of organizing information within a file, depending on what software program created the file. But how the file is organized internally is not important to the disk. What is important is where the file begins, where the middle parts are, and where it ends. In addition, the disk must be concerned with making sure the contents of the file are stored accurately. If the information that created the file began with the letter A, then the first byte of the file should contain an A.Almost everything on a disk is a file of one sort or another. The only things that are not files are some of the structures that the computer uses to manage the disk. These are discussed throughout the rest of this chapter.

Types of Disks

There are basically two kinds of magnetic disks: floppy disks and hard disks. The term "floppy disk" refers to any of the small, removable, relatively low-capacity disks. The common physical sizes for floppy disks are 5.25-inch and 3.5-inch. The term "floppy" stems from the 5.25-inch disks, which are somewhat flexible because they are enclosed in flexible plastic cases. The more recent 3.5-inch disks are enclosed in hard plastic cases. These disks have the advantages of being small enough to fit in a shirt pocket, being more rugged because of their stronger case, and having a higher storage capacity than the 5.25-inch disks.

For each size of floppy disk, there are two densities, termed "low-density" and "high-density." For 5.25-inch disks there are five different capacities supported by DOS 5.0, depending on the number of sides of the disk that can be used and the density of the magnetic particles. For 3.5-inch disks there are three different capacities supported. Both sides of 3.5-inch disks are always usable. The different sizes and capacities are shown in Table 6-1.

Table 6-1 — Floppy Disk Capacities.

Disk Size	No. of Sides	Capacity
5.25"	1	160K
5.25"	1	180K
5.25"	2	320K
5.25"	2	360K
5.25"	2	1.2M
3.5"	2	720K
3.5"	2	1.44M
3.5"	2	2.88M

The 160K, 180K, 320K, and 360K 5.25-inch disks, and the 720K 3.5-inch disk, all have a low density of magnetic particles, and are therefore used with low density drives. The 1.2M 5.25-inch and 1.44M 3.5-inch disks both have a high density of magnetic particles and require high density drives. The 2.88M 3.5-inch disk requires yet another type of drive, specially made to handle its very high capacity. High density disk drives can read and write both low and high density disks, but low density drives can only read and write low density disks. As costs on high density disks come down, low density disks are being used less frequently, and as 3.5-inch disk drives become more common, 5.25-inch disks are being used less frequently.

Unlike floppy disks, which can be easily removed, hard disks are permanently installed in the computer. They have two main advantages over floppy disks: they can store much more information; and they can move information to and from electronic memory much faster. Remember that the computer can only operate on information that has been copied from a disk into memory. Since the computer operates at electronic speeds, the mechanical process of reading from and writing to a disk is the slowest operation the computer performs. Because hard disks are extremely fast at reading and writing, compared to floppy disks, this speed increase alone is enough to justify their existence. Their large capacity is also essential for many of today's programs.

Hard disks come in many sizes. The smallest is 10 megabytes, but hard disks that small are rarely encountered anymore. Sizes today typically range from 20 to 650 megabytes and more.

DOS refers to disk drives by letters of the alphabet. If a computer contains two floppy drives, they are usually referred to as drive A and drive B. If a computer contains a hard drive, that drive is almost always referred to as drive C. If a computer contains only one floppy drive and a hard drive, the floppy drive is generally designated to be both A and B, while the hard drive is still drive C. Additional drives are referred to as D, E, F, etc. When specifying a drive, the letter is followed by a colon. Thus, you type A: to reference drive A.

The Logical Structure of Disks

Disks are physically simple; just platters with a magnetic coating. However, to store information accurately, in a way that the computer can understand and access, requires that a more complex logical structure be imposed on the physical structure.

Tracks, Cylinders, and Sectors

To provide a consistent pattern to where information is stored on a disk, the surface of a disk is divided into concentric circles, called tracks. The number of tracks on a disk depends on what type of disk it is; on a 1.44 meg floppy disk, the surface is divided into 80 tracks. The outermost track is referred to as track 0, and the innermost track is track 79. Thus, on a 1.44 meg floppy, there are 80 concentric circles, starting near the outer edge and ending near the center. The tracks are where information is actually stored on the disk.

To read and write information on a disk, some mechanism must be provided that can locate the tracks and sense the magnetic charge of the particles on them. This mechanism is called a read-write head, or just head for short. The head is attached to an arm that moves back and forth across the disk, positioning the head precisely over each track.

Since there are two sides to a disk, it makes sense to use both sides for storing information. The top side of the disk is side 0, and the bottom side is side 1. Therefore, there are 160 tracks on a 1.44 meg floppy disk — tracks 0 through 79 on side 0, and tracks 0 through 79 on side 1. The read-write head actually consists of two heads, one on each side of the disk. As the disk is used, information is written first to side 0, track 0, then to side 1, track 0, then to side 0, track 1, and so on. It makes sense to use the same track on both sides of the disk before moving on to the next track, because the slowest operation the disk performs is moving the head from one track to another. By going from side 0 to side 1, the head does not have to be repositioned.

When your disk malfunctions, and someone says it needs alignment, what they mean is that the heads are no longer accurately positioning themselves over the tracks. They need to be re-aligned with the tracks, so that they can again read and write accurately.

If you imagine several rings stacked up on top of each other, they make a cylinder. Many hard disks use several disks. This is a relatively inexpensive and compact way to increase the amount of storage provided by a drive, since the only additional parts are the extra platters and heads. DOS treats the entire set of platters as one disk. For each disk, there is a pair of read-write heads attached to a moveable arm. The correspondingly numbered tracks on the two sides of the different disks are referred to together as a "cylinder." Thus, on a 1.44 meg floppy disk, cylinder 0 consists of two tracks: track 0 on side 0, and track 0 on side 1. The number of platters on hard drives varies with the manufacturer and model of drive. Some have just one, while others have several.

On floppy disks, the head actually sits on the magnetic media. This restricts the speed at which the disk can rotate, and therefore the speed with which information can be transferred to and from the disk. On hard disks, the head floats on a thin layer of turbulent air created by the rotating disk. Since this layer is very thin, dust and other small particles could create problems, so they must be kept out. Hard disks are therefore manufactured in "clean rooms" and their assemblies sealed so that no particles can creep in after they leave the clean room. You should never attempt to disassemble a hard disk unit, though you may decide to install or remove the entire unit yourself.

When a disk drive performs a disk access operation (either a read or write), it must access a specific amount of information — it cannot arbitrarily read 10 bytes one time, then 50 the next. The ideal amount of information to be accessed at once would be enough so that most access operations would be accomplished in one step, but not so much that a lot of extra information was accessed. A full track generally contains too much information. To read and write entire tracks would make the disk drive very slow, and would usually require reading or writing much more information than is required by the computer. Tracks are therefore divided into units called "sectors" to provide a sensible size for reading and writing.

Most disks are manufactured to support a variety of sector sizes, so that they can be used with many different types of computers. The actual sector size used is set by the computer's operating system, during a process known as "low-level formatting." All versions of the DOS operating system for IBM-compatible PCs use 512 byte sectors for both hard disks and floppy disks. Sector numbering starts with 1, so a disk that has 18 sectors per track will have sectors numbered from 1 to 18. Sector numbering starts over again for each track. 1.44 meg floppy disks have 18 sectors per track. On a hard disk, the number of sectors per track varies with size and type of disk; 17 sectors per track is common.

It is now apparent how the capacities of disks are determined: the number of bytes per sector, times the number of sectors per track, times the number of tracks, times the number of disk sides, gives the total number of bytes available on a disk. For example, consider the high density 3.5-inch floppy diskette. It has 512 bytes per sector, 18 sectors per track, 80 tracks per side, and two sides. This gives the following calculation:

$$512 \times 18 \times 80 \times 2 = 1{,}474{,}560 \text{ bytes}$$

1,474,560 divided by 1024 (the number of bytes in a kilobyte) gives 1440 kilobytes. Since 1000 kilobytes are referred to as a megabyte (or meg), these disks are referred to as 1.44 meg disks.

Since the number of bytes per sector is fixed at 512, the three variables that allow manufacturers to create disks of different capacities are the number of sectors per track, the number of tracks per disk side (which is the same as the number of cylinders), and the number of physical platters per disk drive (which is the same as the number of disk sides).

Interleave

So far, sectors have been discussed as though they are numbered sequentially around the disk, starting with one and, for a typical 40-megabyte hard drive, ending with 17. When DOS accesses the disk, say to write a new file, it writes to sequentially numbered sectors. Thus, if sectors 9, 10, and 11 are all available, a 1500-byte file will be written on those sectors. But think for a minute how the disk is operating: it is spinning at high speeds, with the head floating over the track. Now consider that DOS can only transfer data in 512-byte chunks — one sector at a time. So it transfers the first 512 bytes to sector 9, then goes back to the electronic memory to fetch the next 512 bytes. But if the computer's transfer time is not fast enough, by the time DOS has retrieved the next 512 bytes from memory and is ready to write to sector 10, that sector has flown by. So now DOS must wait until sector 10 comes around again. This slows down the computer considerably: to write all 17 sectors of a track would require 17 revolutions of the disk.

To get around this problem, sectors are not always numbered sequentially. Instead, the sequence of numbers may proceed to every other sector, so that the physical third sector is sector number two, the physically fifth sector is sector number 3, and so on. Only when the numbering gets back to sector one does the second sector get assigned a number. This scheme is called "interleaving;" when every other sector is numbered sequentially, it is referred to as a "2:1" interleave. If every third sector was numbered sequentially, it would be a "3:1" interleave. Matching the interleave of a drive to the speed at which its computer can transfer data to it is a critical part of designing computers — and one that is not always done correctly.

Disk Performance

When discussing disk drives, particularly hard disks, you will often hear reference made to the "speed" of the disk. This term refers to the "average access time," and is measured in milliseconds. It is one of the critical measurements of a disk drive's performance, and is usually considered as important as the capacity of the disk.

Average access time refers to the amount of time it takes the drive, on the average, to move the head from its current position, to the track you want to access. If the head always started at one edge of the disk, then the average access time would be the time it takes to travel across half of the tracks, since half of the tracks would be less than half way away, and half would be more. If it always started at the middle of the disk, then the average access time would be the time required to move across one quarter of the tracks. However, neither of these numbers is accurate because there is no telling where the head is going to be, before a particular access request. The average access time is actually taken to be the time it takes to move across one third of the tracks. This number bears out in testing as an accurate measure of average access time.

The average access time of hard disks varies tremendously, from less than 10 milliseconds, to over 100. Generally, a time in the 25 millisecond range is considered good.

Disk access is the greatest bottleneck in terms of how fast a computer operates, and moving the head from one track to another is the slowest part of disk access. Anything that can be done to increase disk performance can have significant payoffs for the overall performance of the computer. There are many ways in which both you and the operating system can increase the efficiency with which disk accesses are handled, regardless of the average access time. For example, if files are arranged so that each one is in contiguous sectors on the disk, then less time will be spent moving the head around the disk to locate all the sectors for a particular file. These techniques are discussed in the "Managing Disks" section of this chapter.

Formatting Disks

So far the discussion has centered on relatively low-level characteristics of disks — the size and number of sectors per track, the number of platters per drive, and so on. These characteristics are generally set once and not changed. Setting the number of bytes per sector, the number of sectors per track, and the interleave, are all part of process called "hard formatting," or "low-level formatting." All of this provides the background for the disk to do what it is supposed to do — store and retrieve information. But in order to efficiently store and retrieve information, there must be a way of keeping track of what information is where on the disk — of building the directory structure discussed in Chapter 5. In addition, part of a disk may become damaged, disabling a particular sector. Rather than throw away an entire disk drive because a very tiny percentage of the capacity is disabled, it should be possible to detect and mark those damaged areas so that they are ignored. Setting the disk up for actually handling files and directories, and marking bad sectors, is called a "high-level format." A high level format is what is usually referred to when people speak of formatting a disk.

To provide the necessary logical organization to manage all of the information on a disk, when a disk is formatted DOS divides it into four main areas: the Boot Record, the File Allocation Table (FAT), the Root Directory, and the Data Area. It makes no difference what kind of disk it is, or how large it is. All disks that can be used on a DOS-based computer have these four areas.

The term "Boot Record" comes from the fact that the information stored here is used by the computer during the start-up or "boot" operation. The Boot Record contains a small program that the computer reads when it is started. This program provides the computer with the essential information it needs to continue the boot operation, as well as with information about the disk it is on — the number of sectors per track, the number of sides, and so on. An example of a boot record is shown in Figure 6-1.

```
                   Description              Boot Record Data

OEM ID                              : MSDOS5.0
Bytes per sector                    : 512
Sectors per cluster                 : 4
Reserved sectors at beginning       : 1
FAT Copies                          : 2
Root directory entries              : 512
Total sectors on disk               : (Unused)
Media descriptor byte               : F8 Hex
Sectors per FAT                     : 80
Sectors per track                   : 17
Last side number                    : 5
Special hidden sectors              : 17
Big total number of sectors         : 81923
Physical drive number               : 128
Extended Boot Record Signature: 29 Hex
Volume Serial Number                : 376387808
Volume Label                        :
File System ID                      : FAT16
```

Figure 6-1 —A Typical Boot Record.

This Boot Record indicates, among other things, that the disk it is stored on can contain up to 512 root directory entries, that there are 17 sectors per track, and that the last side number is 5 (which means there are 3 platters, with sides numbered 0 through 5). It also indicates that there are four sectors per cluster on this disk. Recall from Chapter 5 that DOS actually works with clusters as the smallest allocation units on a disk. The number of sectors per cluster varies with different types of disks; floppy disks usually have one or two sectors per cluster, while hard disks typically have four or more. You may have noticed that the free disk space on a disk usually goes down by more than the size of a file that has been added. The reason is that a file may only be one byte in size, but on a disk that has four sectors per cluster, a full cluster, or 2048 bytes, is allocated at once. If you go back to this file and add, say, 1000 bytes, the free disk space will not change because you are only filling up space that has already been assigned to the file.

Since the number of tracks on a disk, and the number of sectors per track, is set during a high-level format, it is possible to adjust these numbers with the FORMAT command, discussed below. However, it is not recommended that you do so.

The Boot Record is always written at the very beginning of a disk on side 0, track 0, sector 1. It is written to every disk that DOS uses, whether those are bootable disks or not. Bootable disks have, in addition to the four areas of the disk discussed here, three special system files copied to them.

To read and write information to disks, DOS must know the status of all the clusters on the disk. It must know whether a cluster is available for use, or whether it has already been assigned to a file. In addition, DOS should be able to keep track of sectors that have been damaged and therefore should not be used. All of this information is stored permanently in the second logical area of the disk — the File Allocation Table, or FAT. The FAT is stored immediately following the Boot Record on a disk.

When you boot your computer the FAT is read into memory. This speeds up the disk performance, since the drive does not have to locate and read the FAT each time you access the disk. However, on a large disk a FAT that kept track of all the sectors on the disk would be quite large. This is why DOS groups sectors into clusters.

The FAT stores a code for every cluster on a disk. On large disks, DOS stores a two-byte code for each cluster, while on floppy disks a 1.5-byte, or 12-bit, code is used. From the value of this code, DOS can determine everything it needs to know about a cluster. If the code is zero, then the cluster is available for use. Assuming a two-byte code, if the code is a number between 2 and 65,519, then the cluster is already in use by a file; in this case, the number points to the next cluster assigned to the file. If the number is between 65,528 and 65,535, then the cluster is assigned to a file, and is also the last cluster for the file.

You may notice there are a few missing numbers in this sequence. If the FAT entry for a cluster is 65,527, then the cluster is bad and should not be used. The numbers between 65,520 and 65,526 are reserved for use in future versions of DOS. The number 1 cannot be used, because the first cluster of the disk is used by the system information: the Boot Record, the FAT, and the Root directory.

Obviously, the FAT is essential to maintaining the integrity of a disk. If a cluster is damaged, you may lose all or part of a file. But if the FAT is damaged, you lose the whole disk. Consequently, DOS actually creates and maintains two identical FATs on each disk. If one is damaged, there is a reasonable chance that the second will be usable.

The Root directory holds the same information as other directories, which were discussed in detail in Chapter 5. However, the Root directory is unique in that it is assigned a specific, fixed location in the system area of the disk. It immediately follows the FAT, and can never grow or shrink. Consequently, the Root directory can only hold a specified number of entries. This number is stored in the Boot Record, and is determined during low-level formatting, and cannot be changed. For the disk in Figure 6-1, there is room for 512 entries in the Root directory. This is a typical value for hard disks.

Among the items of information stored for each file in a directory is the number of the starting cluster for the file. This is why DOS can manage to find files accurately, even if they are highly fragmented and stored in non-contiguous clusters. DOS reads the first cluster number from the directory, then looks in the FAT to find the subsequent clusters. Each cluster points to the next one, until the end of the file is reached.

Following the Root directory is the data area and this occupies the rest of the disk.

Disk Partitions

The last aspect of disk organization that should be reviewed is partitions. Partitioning is a way of taking one physical disk, and making it behave as though it were two or more disks. There are two reasons for doing this. The first is that some people want to use more than one operating system with the computer. Since the formatting of a disk is entirely dependent on the operating system, and every operating system requires disks that were formatted with its own particular needs in mind, the only way to use one hard disk with two operating systems is to partition it.

The second reason for partitioning a hard disk is that all versions of DOS prior to 4.0 could only handle disks that were 32 megabytes or smaller. (Compaq actually changed this with their own version of DOS 3.1, so if you have Compaq

DOS, any version from 3.1 on could produce very large hard drives). The reason for this is that, until DOS 4.0 (or Compaq DOS 3.1), DOS used two bytes to refer to sector numbers on a disk. The largest number two bytes can store is 65,536. Since each sector has to be 512 bytes, this gives 65,536 X 512, or 33,554,432 bytes as the largest possible size for a disk. This is precisely 32 megabytes. Starting with DOS 4.0 (and Compaq DOS 3.1), DOS began using 4 bytes to store a sector number. DOS 5.0 permits partitions of up to 2 gigabytes, or 2,147,483,648 bytes. Some people still prefer to partition a very large hard drive into multiple smaller drives, though it is no longer necessary.

To partition a drive, a partition table is added to the system area of a disk. It was stated earlier that the Boot Record always occupies the very first sector of the first track of side 0 on a disk. This is actually only the case with non-partitionable disks. Partitionable disks, which includes all hard disks but not floppies, place the partition table in the first sector. The partition table is independent of the operating system, so it is set up in the low-level formatting of the disk. The rest of the system area is then bumped forward. Since the computer requires a boot program to be in the first sector of a boot disk, the partition table includes a boot program. All hard disks have partition tables, and technically all hard disks are partitioned, even though some have only one partition that comprises the entire disk.

Managing Disks

Managing disks involves keeping the files that are on them organized so that the disk does not become fragmented, and so that most files occupy clusters that are either contiguous or at least close together on the disk. It also involves configuring your PC to use your hard drive as efficiently as possible.

It is obvious, but bears stating nonetheless, that the most useful thing you can do to keep a disk from running out of space is to delete files that are no longer needed. If there are files you do not want to delete, in case they are needed again in the future, but which are not being actively used, they should be copied to floppies and deleted from the hard disk. Other files to watch out for are temporary files that programs create, and normally delete when the program terminates. These files can be left on the disk if the computer is unexpectedly turned off while the program is running. For example, many word processors create temporary backup files as you work. These are deleted when the word processor is exited, but if the power is shut off while the word processor is still running, these temporary files are left behind. It is worth checking for these files whenever the power fails unexpectedly. Finally, there are several files that come with DOS 5.0 that you may never use. If you will not need one or more of these files, it is worth deleting them to free up disk space. These files are:

EMM386.EXE is an expanded memory manager that can only be used with computers that have a 386 or later CPU. If your computer has a 286 or earlier chip, or if it is a 386 or later but you do not want to use expanded memory, this file can be deleted. This file occupies about 91K of disk space.

EXE2BIN.EXE is a programming tool used to convert files with an EXE extension to a binary file. This is a tool used exclusively by programmers, and even most programmers don't use it. This file occupies about 8K of disk space.

RAMDRIVE.SYS is a device driver used to set up a RAM disk, which is a pseudo-disk in memory. RAM disks can be used to greatly increase the speed at which some programs run, but they can also chew considerable memory resources. If you do not want to set up RAM drives, you can delete this file. This file occupies about 5.5K of disk space. The RAMDRIVE.SYS driver is discussed later in this chapter.

SMARTDRIV.SYS is a device driver used to provide a disk cache for hard disks, on computers that have extended or expanded memory. A disk cache is used to speed up the performance of hard disks, but chews up memory resources in the process. If you do not have extended or expanded memory, or do not want to create a disk cache, you can delete this file. It occupies about 8K of disk space. SMARTDRIV.SYS is discussed later in this chapter.

In addition, the following files are used to provide international standards support. This allows you to change the keyboard to reflect French usage, or to change the internal character set to that used by a foreign language. If you do not need international support, these files can be removed:

File Name	Size
COUNTRY.SYS	16K
DISPLAY.SYS	15K
GRAFTABL.COM	11K
KEBY.COM	14K
KEYBOARD.SYS	34K
NLSFUNC.EXE	7K
PRINTER.SYS	18K
Any DOS file with an extension of CPI	0.4K – 60K

FORMAT

When a disk is new it must be given both a low-level format and a high-level format. The low-level format sets up the number of bytes per sector, the system area, and other details that define how the disk will interact with the operating system. When the term "format" is used with a hard drive, it usually refers just to the high-level format, because the FORMAT command only performs a high-level format on hard disks. When the term is used with floppy disks, however, it means a combined low-level and high-level format because FORMAT performs both of these operations on floppy disks. Hard disks are normally formatted before being sold, while floppy disks need to be formatted after they are purchased.

When a disk is formatted, four things happen: an image of the FAT and Root directory are saved; a new FAT is created; a new Root directory is created; and the disk is checked for bad clusters, which are marked as bad in the FAT. This is sometimes referred to as a "safe" format because it saves an image of the current FAT and Root directory before overwriting them, and it does not affect existing information in the Data Area of the disk. The /U switch, discussed below, can be used to perform an Unconditional format, which does not save an image of the FAT or Root directory. The FORMAT command is used to format a new disk, and to reformat disks that have already been formatted. Reformatting can sometimes correct errors that have crept into a disk.

If you are reformatting a disk that has already been formatted, and you are entirely confident that the disk does not have bad clusters on it, you can use the /Q switch (for Quick). This switch tells FORMAT not to scan the disk for bad clusters, but simply to replace the FAT and Root directory. This is a very quick way to effectively delete all the files on a disk. The /Q switch should never be used to format new disks because new disks sometimes have bad sectors in them and these must be marked in the FAT.

If a disk has given you read and write error messages, reformatting it may solve the problem. In this case, you should use the /U switch. The /U switch tells FORMAT to destroy all information on the disk. This makes it impossible to recover the previous state of the disk with the UNFORMAT command, since no existing information is saved, but it also may eliminate whatever was causing the problems with the disk. The /U switch has the added advantage of being faster, since without it, FORMAT must take the time to save information about the disk. The fastest format, then, is one in which both /Q and /U are used. When formatting new disks, use the /U switch.

FORMAT automatically senses the maximum number of tracks and sectors per track that a floppy drive can handle. It uses those numbers for any disk formatted in that drive, so it is not usually necessary to specify those values. However, it is sometimes desirable to format a disk that has a lower capacity than the maximum the drive can provide; and in some special circumstances, you may want to format a disk that has an unusual number of tracks and/or sectors per track. You can specify a different capacity in one of two ways: with the /F switch (to specify a different DOS capacity), or with the /T and /N switches (to specify the number of tracks and sectors). Using the /F switch is the recommended method. You cannot use the /F switch with either the /T or the /N switch.

The /F switch takes the form /F:capacity, where **capacity** specifies one of the capacities shown in Table 6-2.

Table 6-2 — Values for the FORMAT /F:capacity Switch.

Physical Size	Number of Sides	Storage Size	Capacity
5.25"	1	160K	160
5.25"	1	180K	180
5.25"	2	320K	320
5.25"	2	360K	360
3.5"	2	720K	720
5.25"	2	1.2M	1200
3.5"	2	1.44M	1440
3.5"	2	2.88M	2880

For readability, the size specifiers may have the letter K or the letters KB appended to them. Thus, the following command formats a 3.5" diskette in the A: drive to 720K:

```
FORMAT A: /F:720KB
```

The /F switch sets the capacity of a disk by adjusting the number of tracks, and the number of sectors per track, that the disk uses. Thus, it is an indirect way of adjusting these values. It is the preferred method because it can only use standard DOS values. Furthermore, with /F you don't need to know the number of tracks and sectors, just the capacity you want the formatted disk to have.

The /T and /N switches must be used together. The /T switch specifies the number of tracks, and /N specifies the number of sectors per track. The following command can be used on a high density 3.5" drive (1.44 meg) to format a disk to 720K:

```
FORMAT A: /N:9 /T:80
```

This command creates a disk with nine tracks, and 80 sectors per track, which is the format of a 720K disk. The /F:720 switch would have the same effect.

Caution: When using the /T and /N switches, there is the possibility of formatting a disk to a non-standard number of tracks and sectors per track. This could make the disk unreadable.

Note: When formatting a disk that has been previously formatted, using any switch that changes the capacity of the disk performs an unconditional format, just as though the /U switch had been used. Previously recorded data then becomes unretrievable.

Formatted disks are not automatically "bootable" disks. A bootable disk is one that can be used to boot the computer. For a disk to be bootable, it must have three special system files on it: IO.SYS, MSDOS.SYS, and COMMAND.COM. The first two files have the Hidden and System attributes turned on, so they do not show up in a normal directory listing. They can be seen, however, with any of the following commands:

```
DIR /AH

DIR /AS

ATTRIB *.SYS
```

All three of these files are stored in the root directory of a bootable disk, and contain instructions that the computer needs in order to function properly. To create a bootable disk during the FORMAT operation, the /S switch is used, which copies these files to the disk after formatting is complete. The following command formats a disk in drive A:, and transfers all three system files to it:

```
FORMAT A: /S
```

In all previous versions of DOS, the two hidden system files, IO.SYS and MSDOS.SYS, were required to be located in a particular area of the disk, just as the Boot Record and FAT are. This is not true in DOS 5.0; these files may be located anywhere on the disk. The SYS command, discussed below, is used to transfer these files to a disk that is already formatted. The only requirement is that there be enough free space on the disk to hold them.

The /B switch is a holdover from previous versions of DOS; it is used to reserve the area in which the system files were required to be stored. The system files were not actually transferred to the disk by the /B switch, but the space in which they would be located was marked as reserved, so that DOS would not put other files in that area of the disk. The SYS command could then be used to transfer the system files to the disk at any time. If /B were not specified, then when DOS started allocating space to files it would use the required area. When the SYS command was executed, it would not be able to transfer the files and the disk would not be bootable.

You might wonder why someone would bother to reserve space for the system files, instead of simply sing the /S switch to transfer them during the format. The reason is that the system files, like all commercial software, are copyrighted, and consequently cannot be legally sold or transferred without the owner's permission. Commercial software developers who write programs for PCs sometimes want to sell their software on bootable disks. Since they do not have the legal right to distribute the system files, they would distribute their software on disks that were formatted with the /B switch, then let the customer transfer the files themselves, with the /S switch. It is legal for the customer to do this, as long as they own a legal copy of DOS. The use of /B is no longer necessary, since the system files can go anywhere on a disk formatted with DOS 5.0.

You may specify a volume label for a disk during the FORMAT process by using the /V switch. A volume label may be up to 11 characters, and appears when a DIR command is issued for the disk. The following command formats the disk in drive A:, and gives it a volume label of WP_FILES:

```
FORMAT A: /V:WP_FILES
```

If the /V switch is not used, or if it is used but no label is specified, FORMAT will prompt you for a label. You will still have the choice of leaving it blank. The volume label can always be added or changed later, using the LABEL command.

There are three more switches available for the FORMAT command, all of which are used to deal with older disk technologies that are becoming less and less common. The first is the /1 switch, which is used to format a single-sided floppy disk. The /4 switch is used to format a 5.25" 360K floppy disk in a high density (1.2 meg) drive. Disks that are formatted to low density in a high-density drive are not always readable by low-density drives, so this switch should be used with caution. The /8 switch formats a 5.25" double-density disk to have eight sectors per track (320K). This is the format used by DOS versions prior to 2.0.

DOS keeps track of a value known as the ERRORLEVEL exit code. ERRORLEVEL is called an exit code, because some DOS commands, as well as some software programs, leave a value stored in it when they stop executing. This value can then be used to determine the results of running the program. The FORMAT command sets the ERRORLEVEL code when it terminates; the meaning of each value is shown in Table 6-3.

Table 6-3 — Values of ERRORLEVEL Set by FORMAT Command.

Value	Meaning
0	FORMAT was successful.
3	FORMAT was terminated by the user pressing Ctrl-C or Ctrl-break
4	A fatal error occurred; this number appears for anything that is not covered by values 0, 3, or 5.
5	When the "Proceed with Format (Y/N)?" prompt appeared, the user pressed N.

The ERRORLEVEL variable can be referenced on the IF line of a batch file, which can then take an appropriate action if FORMAT returns a value other than 0. Batch files are discussed in Chapter 8.

UNFORMAT and MIRROR

If a formatted disk is reformatted, or restructured by the RECOVER command, it may be possible to restore it to its condition before the FORMAT or RECOVER command was issued, by using the UNFORMAT command. UNFORMAT will not work on network drives. It can only be used on local hard drives and floppy drives.

To understand how the UNFORMAT command works, it is necessary to have some knowledge of how the MIRROR command and safe formats work. When the MIRROR command is entered, images of the current FAT and Root directory are copied to a file named MIRROR.FIL, in the root directory of the disk. If there is an existing mirror file, it is renamed to MIRROR.BAK, so that you may actually go back two generations in unformatting the disk. The starting cluster for MIRROR.FIL, and MIRROR.BAK if it exists, is then stored in a file named MIRORSAV.FIL. The read-only attribute for MIRROR.FIL and MIRROR.BAK is set, so that these files cannot be modified or deleted, except by the MIRROR program. The hidden, system, and read-only attributes for the MIRORSAV.FIL file are all set, making the file almost invisible and protecting it from deletion. To see this file, enter the command DIR /AH from the root directory. (To delete any of these files, use the ATTRIB command first to change their attributes, then the DEL command to delete them.)

Now, when a safe format is performed, the information in these files remains on the disk. A safe format only replaces the FAT and Root directory; it does not overwrite any of the data area. For most purposes, the files are gone, since the pointers to them in the directory and the FAT are gone, and DOS will overwrite them, because the FAT has all of the clusters in the data area marked as available. But if you know where to look, the data is still there, and the UNFORMAT command knows where to look. This is because the MIRORSAV.FIL file is always written to the last cluster on the disk.

Since UNFORMAT knows where to find the MIRORSAV.FIL file, and MIRORSAV.FIL contains the starting cluster of the MIRROR.FIL and MIRROR.BAK files, those too can be located. So, if the MIRROR command is executed right before safe formatting a disk, UNFORMAT can completely reconstruct the FAT and Root directory. From there, it can attempt to go on and reconstruct the rest of the directory and file structure.

Obviously, if files have been created on a disk after it has been reformatted, they may have overwritten some of the data area that stored files from before the reformat. These files will not be recoverable. Worse yet, they may overwrite the MIRROR.FIL or MIRROR.BAK files. If the disk is filled completely with new files, they will even overwrite MIRORSAV.FIL. Consequently, it is essential that UNFORMAT be used as soon after an accidental reformatting as possible.

If, when the MIRROR program is run, the last cluster is in use, MIRROR will attempt to store MIRORSAVE.FIL in one of the previous four clusters instead. If none of these are available, the following messages appear (assuming drive A is the drive for which the command was entered):

```
Drive A being processed.
Drive A error. Insufficient space for the MIRROR image file.
The MIRROR process was unsuccessful.
```

If you see this message, it may mean that the disk is actually too full, or simply that the last clusters are occupied. If you have less than about 60K of disk space, the former is the case, and you should delete some files. You may still see this message after freeing up enough space, though, if the files you deleted were not stored in the last clusters of the disk. In that case, you should use a disk optimization program. This will rearrange the files on your disk to occupy contiguous clusters, and it will leave all of the empty space on the disk at the end, thereby freeing up the last cluster for MIRORSAV.FIL.

When a safe format is performed, the safe format itself actually creates the same mirror files, so the UNFORMAT command will still work, even if no MIRROR command was executed. These files may be more up to date than the MIRROR files, since they are created immediately before the format operation. If you attempt to perform a safe format on a disk that does not have sufficient space for the mirror files, the following messages appear (assuming that drive A: is being formatted):

```
The MIRROR process was unsuccessful.
Checking existing disk format.
Saving UNFORMAT information.

Drive A error. Insufficient space for the MIRROR image file.
There was an error creating the format recovery file.
This disk cannot be unformatted.
Proceed with Format (Y/N)?
```

If the MIRROR command is used on a hard disk, and an unconditional format is then performed on that disk, UNFORMAT may still work, because an unconditional format does not destroy the data area of a hard disk. It only does this with a floppy disk.

If an unconditional format is used on a hard disk that does not have mirror files saved on it, UNFORMAT will still attempt to reconstruct the disk as best it can. It does this by searching every cluster on the disk for information that is stored there, and attempting to deduce what type of information it is. For example, it can usually identify subdirectories because of their unique structure. Once a subdirectory is identified, it can be used to identify the files it contained, and the starting clusters for those files. The less fragmented a disk, the better the chances UNFORMAT has for recovering files, since files that are unfragmented are easier to reconstruct. This is a very slow process, since every part of the disk must be read and analyzed.

To summarize, then, UNFORMAT will work under the following circumstances:

1. Either a hard disk or a floppy disk was safe formatted, without the /U switch, and without changing its capacity if it was a floppy disk. If no changes were made to the disk following the safe format, it should be recoverable. Depending on the magnitude of changes made after the format, some or all of the previous information may be lost.

2. The MIRROR command was used on a hard disk, which was then unconditionally formatted (the /U switch was used). In this case, UNFORMAT will attempt to restore the disk to its condition at the time the MIRROR command was used. Subsequent changes will be lost, and if subsequent changes were made to the disk, some of the pre-format information may be lost as well.

3. A hard disk was unconditionally formatted and the MIRROR command was not used. In this case, UNFORMAT attempts to reconstruct the disk from information in the data area alone. This process is very slow, and is the least likely to produce satisfactory results, since UNFORMAT must attempt to reconstruct the disk from the information in the data area alone.

UNFORMAT will be completely useless with a floppy disk that has been unconditionally formatted, and with any disk that has been completely filled up with new files after any type of format.

To unformat a disk, enter the following command:

```
UNFORMAT X:
```

where **X** is the drive letter for the disk to be unformatted.

If you are using the MIRROR program to protect your disk, you can use the UNFORMAT command with the /J switch to determine whether the mirror file is current. When /J is used, it first checks to make sure the disk has a valid mirror file. If it does, the message "The MIRROR image file has been validated." is displayed, and UNFORMAT goes on to compare the FAT in the mirror file with the actual FAT on the disk. If no changes have been made to the disk since the mirror file was created, the message "The system area of drive A has been verified to agree with the MIRROR image file." will appear. If changes have been made, the message "The system area does not agree with the MIRROR image file." appears. This lets you know that you should run the MIRROR program again. MIRROR should be run frequently on disks that are used regularly, to keep the mirror files as up to date as possible. Putting the MIRROR command into the AUTOEXEC.BAT file is a good way to ensure that this is done.

UNFORMAT uses a mirror file if one is present. This is usually desirable, but if the mirror file is very old, UNFORMAT may do a better job without it. There is no way to delete the mirror file, since as far as DOS is concerned it does not exist, so the /U switch is used to tell UNFORMAT to ignore the mirror file.

Before using /U, it is a good idea to run UNFORMAT with the /TEST switch. /TEST simulates the unformatting of the disk, without actually carrying it out. It can be a little unnerving to use this switch, because when it starts, you are asked if you are sure you want to do this, as though it will actually carry out the process. However, after you respond with a Y, the message "Simulation only." appears. You can treat further warnings about proceeding in the same vein. Be aware that both the /J and /TEST are very slow processes, so allow plenty of time for them. /TEST may be combined with the /U, /L, /P, and /PARTN switches.

Normally, when UNFORMAT is rebuilding a disk without using a MIRROR file, it displays the names of subdirectories it finds on the disk, and the names of files that it finds are fragmented on the disk. The reason it displays fragmented files is that it cannot restore them completely without a MIRROR file, and therefore prompts you to indicate whether it should recover the first part of the file, and truncate it at the point of fragmentation, or not recover any of it. This is why it is wise to run a disk optimization program frequently. Files that UNFORMAT does not prompt you for are ones that it perceives as intact, and therefore can be completely recovered. However, it is possible that UNFORMAT will occasionally believe that it has recovered a file completely, when in fact the file is fragmented and therefore has not been completely recovered. In these cases, when you discover that the file is incomplete, it must be restored from a backup copy or, in the case of program disks, from the original disks.

The /L switch performs two functions: it causes UNFORMAT to attempt to reconstruct the disk without using the MIRROR file; and it tells UNFORMAT to list all files it finds, regardless of whether or not they are fragmented.

The /P switch sends all output messages to the printer. As there are a number of messages that the UNFORMAT command can display, depending on switch settings and what it encounters along the way, this can provide a useful record of what happened. It is particularly helpful with the /TEST switch, since /TEST is used specifically to study the potential results of using UNFORMAT.

When a MIRROR file is found by UNFORMAT, a message indicating the date and time of its creation is displayed. But if a MIRROR file already exists when the MIRROR command is used, the current MIRROR file is renamed to MIRROR.BAK. Thus, there could be two MIRROR files on the disk. If this is the case, the following messages will appear with the UNFORMAT command:

```
The last time the MIRROR or FORMAT command was used was at 17:19
on 05-01-91.
The prior time the MIRROR or FORMAT command was used was at 17:14
on 05-01-91.

If you wish to use the last file as indicated
above, press L. If you wish to use the prior
file as indicated above, press P. Press ESC
to cancel UNFORMAT.
```

Pressing L causes UNFORMAT to use the most recent MIRROR file, while pressing P causes it to use the backup MIRROR file. Normally, you would never use the previous MIRROR file, since fewer changes will have occurred since the most recent MIRROR file was created. However, if you create a MIRROR file, and the disk becomes corrupted, and then you use MIRROR again, then format the disk, the recent MIRROR file may not be usable. In this case, the previous MIRROR file should be used.

The MIRROR and UNFORMAT commands can also be used to protect the partition table of a hard disk. If the hard disk is damaged and the partition table becomes corrupted, these commands can be used to recover it. The technique here is identical to that for saving the FAT and root directory: a copy of the partition table is made in a file named PARTNSAV.FIL. However, because a corrupted partition table will make the hard disk impossible to access, the PARTNSAV.FIL file must be stored on a floppy disk. It is a good idea to make this floppy disk bootable, since you won't be able to boot your computer from the hard drive. This disk can then be used to boot the computer and the UNFORMAT command can be used to restore the partition table. The following MIRROR command will create the PARTNSAV.FIL file:

```
MIRROR /PARTN
```

The MIRROR program will display the following messages:

```
Disk Partition Table saver.

The partition information from your hard drive(s) has been read.

Next, the file PARTNSAV.FIL will be written to a floppy disk. Please
insert a formatted diskette and type the name of the diskette drive.
What drive? A
```

You may either accept the suggested drive letter or enter a different one. MIRROR then goes ahead and creates the PARTNSAV.FIL file on the floppy disk. The UNFORMAT command may now be used with the /PARTN switch to rebuild a damaged partition table. When UNFORMAT is used with the /PARTN switch, UNFORMAT prompts you to put the disk with the PARTNSAV.FIL file in a floppy drive and press Enter←. UNFORMAT compares the detailed specifications of the drive being restored with the information in PARTNSAV.FIL. If the information does not match precisely, UNFORMAT will terminate the operation. If it does match, UNFORMAT goes ahead and restores the partition table. After restoring the partition table, you will probably need to use UNFORMAT without the /PARTN switch to recover the files on your disk.

When the /L switch is specified with the /PARTN switch, it displays the information in the partition table of the current drive. Nothing on the disk is modified, so this can be done at any time. An example of how the partition table appears with this command is shown in Figure 6-2.

```
Hard Disk Partition Table display.

Drive # 80h has 976 cylinders, 5 heads, 17 sectors (from BIOS).

The following table is from drive 80h, cylinder 0, head 0, sector 1:

               Total_size        Start_partition    End_partition
   Type       Bytes   Sectors    Cyl Head Sector    Cyl Head Sector      Rel#
-----------  -----------------  ----------------  ----------------  ----------
HUGE  Boot    40M    81923       0    1     1    963    4    17            17
```

Figure 6-2 — Partition Table Displayed with UNFORMAT /PARTN /L.

Partitions are created on hard disks using the FDISK command, discussed in the next section.

UNFORMAT and MIRROR are external commands.

FDISK — Partitioning a Hard Drive

Partitioning a disk is the process of dividing the disk into distinct units that are each treated as though they were separate drives. Only hard disks may be partitioned. The reasons for partitioning a hard disk were discussed earlier in this chapter.

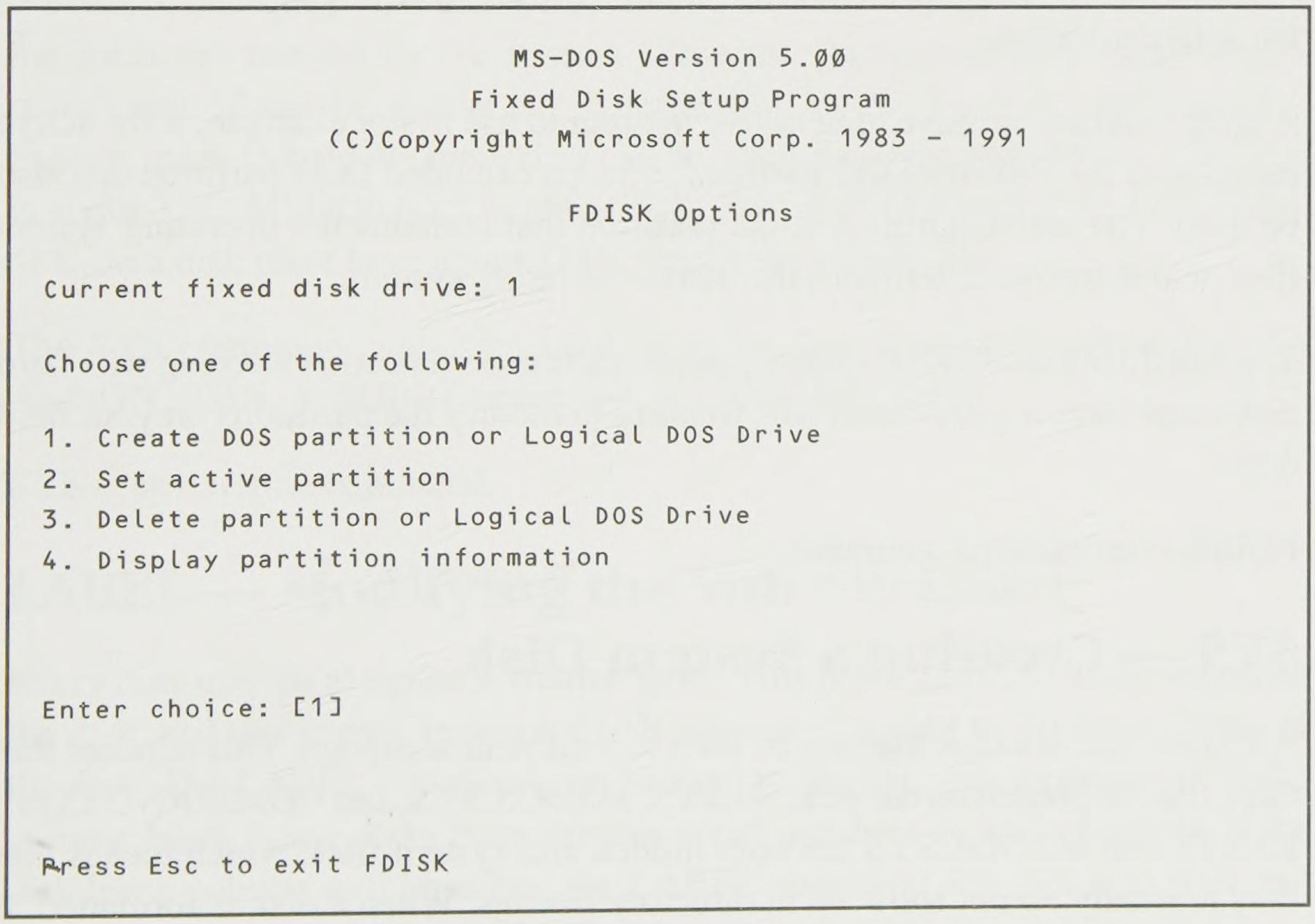
WARNING

The FDISK command is used to partition a drive. This command should only be used if you are quite confident of what you are doing. When it is used to change a partition on a disk, all of the information in the modified partition is lost. You should always back up your disk before using the FDISK command for anything other than viewing the current partition table.

There are three types of partitions that a disk can have: a primary DOS partition; an extended DOS partition; and non-DOS partitions. Non-DOS partitions are for use by operating systems other than DOS, such as Unix or Xenix. With DOS 5.0, you may have two DOS partitions on a single hard drive: one primary partition, and one extended partition. The primary DOS partition is a partition that contains the system files required for booting the computer — IO.SYS, MSDOS.SYS, and COMMAND.COM.

The extended DOS partition may be subdivided into multiple logical drives, or may be used as a single drive. Consider a 120-megabyte hard drive. Using FDISK, you could create a primary DOS partition that is 40 megabytes in size, and is referenced as the C: disk. This would be the logical disk that the computer would boot from. The other 80 megabytes would then be used to create an extended partition. In order to use the extended partition, one or more logical drives must be created on it. If you create more than one logical drive, the sum of the drive sizes should total the size of the extended partition.

When the FDISK command is entered, the screen shown in Figure 6-3 will be displayed.



```
                         MS-DOS Version 5.00
                       Fixed Disk Setup Program
                (C)Copyright Microsoft Corp. 1983 - 1991

                             FDISK Options

Current fixed disk drive: 1

Choose one of the following:

1. Create DOS partition or Logical DOS Drive
2. Set active partition
3. Delete partition or Logical DOS Drive
4. Display partition information


Enter choice: [1]



Press Esc to exit FDISK
```

Figure 6-3 — The FDISK Menu.

Option 4 in Figure 6-3 displays the current partition information. If you choose one of the other options, remember that modifying any existing partition will destroy the information in it.

It is not possible to simply change a partition by enlarging or shrinking it. Rather, you must delete it and create a new partition with the characteristics you desire.

Option 3 refers to the "active" partition. When a disk has partitions for more than one operating system, each operating system must have one partition that contains its system files. To use the partition(s) that are compatible with that operating system, the partition that has the system files must be active. For example, suppose you had both DOS and Unix operating systems on your computer. Each of these would have a partition that contained their operating system files. For DOS, this would be the Primary DOS partition. If this was assigned to be the active partition, then when the computer was started it would boot with the DOS operating system. If you wanted to boot with the Unix operating system, you would have to change the active partition.

A partition does not have to be active in order to use it. For example, if the active partition is the Primary DOS partition, then an extended DOS partition can also be used. The active partition is the partition that contains the operating system files, and therefore determines the active operating system.

You should not use FDISK unless you are quite experienced in working with hard disks, and have a good reason for wanting to modify the partitions on your hard drive.

FDISK is an external command.

SYS — Creating a System Disk

A system disk is a disk that can be used to boot your computer. This requires that three files be present on the disk: IO.SYS, MSDOS.SYS, and COMMAND.COM. IO.SYS and MSDOS.SYS are both hidden and system files, which means that they normally do not show up in directory listings. When a disk is formatted, it can be made a system disk by including the /S switch. If a disk was formatted without /S, and you now want to make it a system disk, use the SYS command.

The system files are always placed in the root directory. To create a system disk, the following command copies the files from the root directory of the current disk to the disk in the A: drive:

```
SYS A:
```

If the current drive does not contain a system disk, another source drive may be specified. Regardless of what drive is current, the following command could be used to transfer the system files from the root directory of the C: drive to the disk in the A: drive:

```
SYS C:\ A:
```

SYS copies the files in the following order: IO.SYS, MSDOS.SYS, and COMMAND.COM. In previous versions of DOS, the IO.SYS and MSDOS.SYS files had to be located in specific positions near the beginning of the data area on the disk. Consequently, if a disk had files already on it, those files would occupy the locations needed by the system files, and the system files could not be transferred. However, this is not the case with DOS 5.0, so any disk that has enough space to hold the three files can be made a system disk. The IO.SYS file is about 33K, MSDOS.SYS is about 37K, and the COMMAND.COM is about 48K, so a disk must have about 118K free to store these files.

The SYS command cannot be used on drives that have been affected by the ASSIGN, JOIN, or SUBST command, or on network drives.

SYS is an external command.

LABEL — Modifying the Volume Label

Every disk may be assigned a volume label. This is a name that can be stored on the disk, and that shows up when a DIR command is used to list the contents of the disk. The LABEL command can be used to display, add, or replace a disk's volume label. Some disks have volume serial numbers as well as labels. If the disk has a volume serial number, the LABEL command displays it as well, but will not allow you to change it. Serial numbers are assigned when disks are formatted by any version of DOS from 4.0 on.

If no drive is specified on the command line, the LABEL command applies to the current drive. The following two commands demonstrate this: the first changes the label of the disk in the current drive to TEST_DISK, while the second changes the label of the disk in the A: drive to TEST_DISK:

```
LABEL TEST_DISK

LABEL A: TEST_DISK
```

If no label is specified with the LABEL command, it displays the current label, and the volume serial number if there is one, and prompts you for a new label with the following messages:

```
Volume in drive A is TEST_DISK

Volume Serial Number is 112A-18E4

Volume label (11 characters, ENTER for none)?
```

If there is no volume serial number, that part of the message is not displayed. Entering a new label and pressing Enter↵ replaces the label. If you press Enter↵ without entering a new label, the following message appears:

```
Delete current volume label (Y/N)?
```

Responding with a Y removes the current label, leaving the disk with no volume label. Respond with N leaves the current label intact.

LABEL is an external command.

VOL — Displaying the Volume Label

The VOL command is used to display the volume label and, if it exists, the volume serial number. If no drive is specified, the current drive is assumed. The following command displays the volume label and serial number of the disk in drive A:

```
VOL A:
```

VOL is an internal command.

CHKDSK — Checking Disk Integrity

The CHKDSK command is used to display information about the capacity and available space on a disk and on the status of memory usage, and to check the integrity of the FAT and directory structure of a disk.

Recall that the FAT keeps track of the status of every cluster on the disk. Each cluster can be labeled as available for use, as in use by a file, or as defective. It sometimes happens that an error creeps into the FAT, leaving a cluster marked as in use, when in fact it is no longer part of a file. This can be determined by studying the starting clusters for files, as listed in the directories, and following the chains that start with those clusters to the end of the file. A cluster that does not fall into one of these chains is lost; it cannot be assigned to an new file, and it does not belong to an existing file. CHKDSK refers to these as "lost allocation units" because a cluster is DOS's unit of allocation — disk space is allocated one cluster at a time.

To run CHKDSK for the current drive, just type CHKDSK and press Enter←. To run it for another drive, append the drive letter to the command. Running CHKDSK for the C: drive produces a report similar to that shown in Figure 6-4.

```
Volume Serial Number is 166D-4E46

  33454080 bytes total disk space
    131072 bytes in 6 hidden files
    126976 bytes in 57 directories
  29497344 bytes in 1116 user files
   3698688 bytes available on disk

      2048 bytes in each allocation unit
     16335 total allocation units on disk
      1806 available allocation units on disk

    654336 total bytes memory
    385952 bytes free
```

Figure 6-4 — Output From CHKDSK Command.

If there had been lost clusters on the disk, CHDKSK would have displayed the following messages:

```
XX lost allocation units found in YY chains.
Convert lost chains to files?
```

Where **XX** is a number that indicates how many lost clusters were located, and **YY** indicates that some of those clusters are chained together, which means they once belonged to the same file. For example, suppose a file occupies clusters 10, 15, and 20, in that order. The directory entry for the file will indicate that the first cluster in the file is cluster 10. The FAT entry for cluster 10 will point to cluster 15, and the FAT entry for cluster 15 will point to cluster 20. The FAT entry for cluster 20 will be marked as the end of the chain. Now suppose you delete the file, so the directory entry is marked as deleted, but something goes wrong so that only cluster 10 is marked as deleted in the FAT. Clusters 15 and 20 continue to be marked as occupied, but they no longer belong to a file. Since cluster 15 points to cluster 20, and cluster 20 is marked as the end of the file, CHKDSK would report 2 lost allocation units in one chain.

Lost clusters can result from a bug in a program, or a power failure while a file is being written, or some other problem. A lost cluster may contain important data. CHKDSK can be used to recover these clusters and chains, so that you may examine them and decide whether or not to keep them. In the messages shown above, CHKDSK prompts you as to whether or not it should convert lost chains to files.

STOP

This can be a misleading question, because answering Y only works if the /F switch was used with CHKDSK. If /F was not used, then CHKDSK takes no action, even if you press Y. However, if /F was used, pressing Y will cause CHKDSK to convert each chain of lost clusters to a file in the root directory. The names of the files will be FILExxxx.CHK, where **xxxx** is a number. The first file converted is FILE0001.CHK, the second is FILE0002.CHK, and so on.

When a program needs to access a file, the file must first be "opened." When a file is opened, DOS assigns it a number, known as a file handle, and tells the program what the file handle number is. The program then refers to the file by its handle. When DOS closes a file, it releases the handle so it can be used by the next file.

Normally, there will be no open files when you run CHKDSK, because it is run at the command prompt. However, if you are running a multi-tasking environment like DESQview or Windows, then there may be open files. CHKDSK should not be used in these cases, because the presence of open files may result in erroneous reporting of lost allocation units.

If CHKDSK reports a large number of lost cluster, you may want to consider having your disk checked for servicing. Lost clusters should be a relatively rare occurrence.

If CHKDSK reports bad sectors on the disk, this is because it detected clusters that were marked as bad in the FAT. This is not something you need to worry about, since they have been marked in the FAT and they will not be used. This information is provided simply as one of the statistics about the disk.

You can save the output from the CHKDSK command to a file by using the output redirection symbol (>), but you should not do so when using the /F switch. The following command performs a CHKDSK on the current drive, and stores the output in a file named CHECK.DAT:

```
CHKDSK > CHECK.DAT
```

The /V switch can be used to tell CHKDSK to display each file name on the disk as it checks the files.

Note that CHKDSK only checks for logical errors in the FAT. It does not check the physical integrity of the disk, nor does it recover files from a physically damaged disk. The RECOVER command must be used if the disk is physically damaged.

CHKDSK cannot be used on drives affected by the ASSIGN, JOIN, or SUBST commands, or on network drives.

CHKDSK is an external command.

DISKCOMP — Comparing Two Floppies

DISKCOMP compares the exact contents of two floppy disks. The two disks must be the same size and format. DISKCOMP performs a track-by-track comparison, so it is not sufficient for two disks to contain the same set of files. The files must occupy the exact same clusters on each disk. It is useful to perform a DISKCOMP after performing a DISKCOPY command, to ensure that the DISKCOPY accurately reproduced the source disk.

The following command compares the disk in drive A: with the disk in drive B:

```
DISKCOMP A: B:
```

There are two switches that may be used with DISKCOMP: /1 and /8. The /1 switch instructs DISKCOMP to compare only the first side of the disks, even if they are two-sided disks. The /8 switch tells DISKCOMP to compare only the first 8 sectors per track, regardless of how many sectors the disks have.

If the two disks are identical, DISKCOMP displays the message "Compare OK." If they are not, it displays a message indicating the side and track on which the mismatch occurred.

You may perform a DISKCOMP, using only one drive by specifying the same drive letter for both the first and second drives. DISKCOMP will then prompt you to put the first and second disks in the drive as it needs them.

The DISKCOMP command sets the ERRORLEVEL exit code when it terminates and the meaning of each value is shown in Table 6-4.

Table 6-4 — Values of ERRORLEVEL Set by DISKCOMP Command.

Value	Meaning
0	The two disk are identical.
1	The two disks are not identical.
2	DISKCOMP was interrupted by the user pressing Ctrl-C or Ctrl-break.
3	A hard error occurred.
4	An initialization error occurred.

DISKCOMP is an external command.

APPEND

The APPEND command is similar to the PATH command, only it operates on data files instead of executable files. Recall that the PATH sets up a list of directories in which DOS will search for executable files, when a command is entered that is neither an internal DOS command, nor a program in the current directory. The APPEND command sets up a list of directories that DOS will search for data files that programs attempt to open, if those files are not found in the current directory. The command is called APPEND because for the purpose of looking for data files, these directories are treated as if they were appended to the current directory. The DIR command does not show files in appended directories.

You may use different drives as well as different directories with the APPEND command. When you are appending two or more directories, they should be separated by a semicolon. For example, the following command appends the root directory in drive A: and the \LETTERS directory in drive B: to the current directory:

```
APPEND A:\;B:\LETTERS
```

If APPEND is entered by itself, it will display the list of currently appended directories. To cancel the effect of the APPEND command, enter the command with a semicolon:

```
APPEND ;
```

When you specify a data file in a program, you may usually include a path if the file is not in the current directory. This normally does not disable the effect of APPEND — if the file is not found in the specified directory, DOS will go ahead and search the appended directories for it. However, you may want DOS to ignore appended directories when a path is included with the file specification. This can be accomplished with the /PATH switch. When /PATH:ON is specified, DOS searches the appended directories, regardless of whether a path is specified with the file name. When /PATH:OFF is specified, DOS only searches appended directories when there is no path specified with the file name.

Normally, DOS searches the path for program files and appended directories for data files. However, you can instruct DOS to use the appended directories as though they are part of the path — to search them for programs as well as for data files — by using the /X switch. If you want to do this, you must use /X:ON the first time APPEND is used after booting your computer. You can then specify /X:OFF to turn APPEND off, and /X:ON to turn it back on. If /X:ON is not specified the first time APPEND is used, X:ON and X:OFF will have no effect. /X:ON can be abbreviated to just /X.

You may store the list of appended directories in a DOS environment variable named APPEND by using the /E switch. Programs that are designed to do so can then access this variable, to see what directories are appended. The /E switch must be used by itself the first time you use APPEND after booting the computer, followed by an APPEND command that specifies the appended directories. You cannot specify appended directories on the same command line as the /E switch. The following two commands create the environment variable, and append the A:\ and B:\LETTERS directories to the current directory:

```
APPEND /E
APPEND A:\;B:\LETTERS
```

Note: APPEND should only be used with data files that will not be copied by a program. For example, suppose your word processor saves a file by copying a new file with the same name. If the file that you are working on is in an appended directory, then when it is saved, a new file of the same name will be created in the current directory, since APPEND does not affect the location of files being created. This leaves two files of the same name on the drive, but you will be unaware that this has happened. This can result in confusion, and in the wrong file being used at a later time. If the program you are using modifies a file without replacing it, then it is okay to use it with appended directories. Database programs and spreadsheets generally modify files without making new copies, while word processors often create new copies of files when they are saved.

APPEND is an external command.

ASSIGN — Reassigning Drive Letters

Some software, particularly older programs, insists on disks being in a certain drive, usually A: or B:. However, you may want the disk in some other drive. One of the most common occurrences is when software comes on a 5.25-inch disk, and insists that the disk be in the A: drive. Many PCs today have a 3.5-inch drive as the A: drive, and a 5.25-inch drive as the B: drive. The ASSIGN command gets around this problem by tricking DOS into thinking that a reference to one drive letter is really a reference to another drive letter. The following command assigns drive B: to drive A:

```
ASSIGN A=B
```

The effect of this command is that whenever a program references either the A: drive or the B: drive, DOS will interpret it as a reference to the A: drive. This can be demonstrated by performing a DIR command on drive B: — DOS will display the directory of the disk in the A: drive.

You may enter colons after the drive letters if you wish, but they are optional.

To clear all assignments, and restore the original meaning of drive letters, enter the ASSIGN command by itself, with no drive letters. To see the current assignments, enter the following command:

```
ASSIGN /STATUS
```

The /STATUS switch may be abbreviated to /STA, or just to /S.

Any given drive letter can only have one other drive letter assigned to it at a time. For example, if you enter the command ASSIGN A=C, then later enter the command ASSIGN B=C, after the second command, drive A will no longer be assigned to drive C. You may, however, use the SUBST command to substitute the C drive for the A drive, and still use the ASSIGN command to reassign B to C.

If you want to use both the ASSIGN and APPEND commands, then the APPEND command must be entered before the ASSIGN command, even if ASSIGN and APPEND are referencing different drive letters.

You cannot assign a hard disk to another drive, though you can assign a floppy drive to a hard drive. Also, you may not use a drive letter that does not exist, as you can with the SUBST command. Furthermore, although it may work, it is best not to use ASSIGN with the BACKUP, JOIN, LABEL, RESTORE, and SUBST commands. The DISKCOPY and FORMAT commands ignore drive reassignments. ASSIGN may be used with network drives.

The SUBST command is preferred over ASSIGN, and should be used instead whenever possible. The ASSIGN command may be dropped from future versions of DOS.

ASSIGN is an external command.

JOIN — Joining Two Drives

The JOIN command makes a drive behave as if it were a subdirectory of another drive. For example, the following command makes the A: drive behave as if it were the C:\WP\TEMP directory:

```
JOIN A: C:\WP\TEMP
```

Now, when the C:\WP\TEMP directory is referenced, the A: drive will be accessed. When this JOIN command is entered, the \WP\TEMP directory must either not exist, or be an empty directory, on the C: drive. If the \WP\TEMP directory does not exist, the JOIN command creates it. Any legal directory name could be used in place of \WP\TEMP, as long it is either empty or non-existent. The requirement that the directory be empty makes sense; if there were files in the directory, DOS would have no way of knowing whether you wanted to reference the files in the real directory, or on the joined drive. If you attempt to create a JOIN with a non-empty directory, JOIN displays the message "Directory not empty," and does not carry out the JOIN.

The drive that is joined to a directory on another drive becomes invalid. In the example above, if an attempt was made to reference the A: drive, the message "Invalid drive specification" would be shown.

If you want to see a list of the currently joined drives and directories, enter the command JOIN by itself. For each joined drive, a message similar to the following will be displayed:

```
A: => C:\WP\TEMP
```

A JOIN may be deleted with the /D switch. When /D is used, only the drive letter is specified, not the joined directory. The following command removes the JOIN defined above:

```
JOIN A: /D
```

There are several DOS commands that do not work with directories formed by the JOIN command: BACKUP, CHKDSK, DISKCOMP, DISKCOPY, FDISK, FORMAT, LABEL, MIRROR, RECOVER, RESTORE, and SYS.

JOIN is an external command.

SUBST — Substituting Drive Letters

The SUBST command is similar to the ASSIGN command, but is more flexible. SUBST allows you to assign a subdirectory path from any real drive to another real drive, or to a "virtual" drive — a legal drive letter for which there is no physical drive on the computer. SUBST should be used in place of ASSIGN whenever possible, as the ASSIGN command may be dropped from future versions of DOS.

To substitute the root directory of the C: drive for the A: drive, enter the following command:

```
SUBST A: C:\
```

This is equivalent to the command:

```
ASSIGN A=C
```

However, SUBST will also work with this command:

```
SUBST A: C:\WP\LETTERS
```

Now, references to drive A: will be translated into references to the \WP\LETTERS directory of the C: drive. Furthermore, you may substitute a drive that does not exist for a subdirectory on one that does exist. For example, the following command substitutes the drive letter E: for the C:\WP directory:

```
SUBST E: C:\WP
```

Now, references to drive E: will be interpreted as references to the \WP directory of the C: drive. With this substitution in place, the following command would display the files in the C:\WP\LETTERS directory:

```
DIR E:\LETTERS
```

The drive E: in this example is referred to as a "virtual" drive, since there is no physical drive labeled E:. Virtual drives may use any letter of the alphabet that is not already assigned to a physical drive. However, they must be in the range specified by the LASTDRIVE command in your CONFIG.SYS file. The LASTDRIVE command is discussed in Chapter 9.

A drive substitution may be deleted with the /D switch. The following command deletes the above substitution:

```
SUBST E: /D
```

If you want to list the current substitutions, enter the SUBST command by itself.

The commands listed in Table 6-5 cannot be used with drives affected by SUBST.

SUBST is an external command.

BUFFERS

When DOS reads information from a disk, it reads at least one sector of information at a time. For example, if you request that a database retrieve the name of a person — say John Smith — DOS will not only read the name John Smith from the disk, but will also read the rest of the information that is in the sector that stores the name on the disk. This is done intentionally because disk reads, even with a hard disk, are the slowest operation your computer has to carry out. By reading in extra information, DOS minimizes the number of disk accesses it must carry out. The information that DOS reads from the disk is stored in areas of memory that are set up when you boot the computer. These areas are known as buffers.

The next time DOS receives a request to read information from the disk, it will first look in the buffers to see if the information is there. If it is, DOS reads it from the buffers and saves the time required to physically access the disk. There is a reasonably good chance the information will be in a buffer, since it is common for two disk reads to be retrieving related information, and related pieces of information are often stored near each other on the disk. If you pay attention to your disk drive indicator light you will notice that it does not always go on when you request information from the disk. When it does not, DOS is finding the requested information from the buffer. You will also notice that information retrieved from a buffer appears on the screen more quickly than information that requires a disk access. DOS will also use the buffers when writing to a disk, which further minimizes disk accesses.

DOS can have more than one memory buffer for storing information from the disk. Each buffer can hold one 512-byte disk sector. The more buffers DOS has, the more likely it is that the needed information will be in memory, thus avoiding additional disk accesses. However, as the number of buffers goes up so does the amount of memory required for them. This memory is permanently set aside for these buffers, so it is taken away from the memory available to programs.

The number of buffers that DOS will set up is determined by the BUFFERS command in the CONFIG.SYS file. This number cannot be changed after the computer is booted, except by modifying the CONFIG.SYS file and rebooting the computer. The following line in the CONFIG.SYS file instructs DOS to set up 15 buffers:

```
BUFFERS=15
```

You may specify anywhere from one to 99 buffers. If you do not include a BUFFERS command in your CONFIG.SYS file, DOS will use a default value based on the configuration of your computer. The default values for the various configurations are shown in Table 6-5.

Table 6-5 — Default Number of Disk Buffers.

Default Number Of Buffers	Condition
2	Computer has no more than 128K of RAM, and no disk drive greater than 360K.
3	Computer has no more than 128K of RAM, and a disk drive greater than 360K of RAM.
5	Computer has between 128K and 255K of RAM.
10	Computer has between 256K and 511K of RAM.
15	Computer has more than 512K RAM.

There is one additional parameter that may be used with the BUFFERS command. This parameter specifies the number of "look-ahead buffers," also referred to as the "secondary buffer cache." A look-ahead buffer is used to read more than one sector from the disk when a disk read is performed. When look-ahead buffers are specified, DOS not only reads the sector that contains the information requested into a buffer, but also reads the following sectors into buffers. The number of additional sectors read is specified by the look-ahead buffer parameter. The following command tells DOS to set up 25 regular buffers and 8 look-ahead buffers:

```
BUFFERS = 25,8
```

You may specify anywhere from 1 to 8 look-ahead buffers. If you do not specify look-ahead buffers, the default value is one. Look-ahead buffers tend to speed up word processors and computer language compilers more than databases and spreadsheets, because the information needed by word processors and compilers is more likely to be stored in adjacent disk clusters.

Each buffer consumes about 532 bytes of memory. Thus, 10 buffers consumes a little over 5K. The trick in setting the number of buffers is to find a number that provides the maximum performance increase, but is not so large that some of the buffer space is never used effectively. The default values for buffers are based on the amount of memory in your computer, but in fact a more important consideration is probably the size of your hard disk — the larger the disk, the more buffers you are likely to need. Table 6-6 provides guidelines for setting the BUFFERS command based on the capacity of your hard disk.

Table 6-6 — Recommended BUFFERS Settings for Various Hard Disk Capacities.

Hard Disk Capacity	Recommended BUFFERS Setting
Under 40 megabytes	20 - 25
40 to 80 megabytes	30 - 35
80 to 120 megabytes	40 - 45
More than 120 megabytes	50 - 60

Another factor in considering how to set your BUFFERS command is the type of software you will use. If you use a database system or another application that reads and writes large amounts of information to and from the disk it may help to increase the number of buffers. This is because these systems frequently access different sectors of the disk on each read or write, so having more buffers available increases the likelihood that a requested sector will be in memory. For most database applications setting 20 buffers is a good place to start.

Some programs require that a certain number of buffers be specified in order to run properly. The number specified is always the MINIMUM number required, so if one program you use requires 15 buffers, and another program requires 20, then 20 will work for both programs.

If you have a performance problem that you believe may be alleviated by adjusting the BUFFERS command the best thing to do is to try out different values and see how they affect the problem. For example, if your database seems to be performing poorly it could be because you do not have enough buffers available, resulting in more disk accesses than necessary. It could also be because you have too many buffers specified, not leaving enough memory for the database application to efficiently perform its processing, in which case it will have to use the disk for tasks it would otherwise use memory for. The only way to determine whether adjusting the BUFFERS command will help is to try different numbers of buffers and see what effect this has.

If your computer takes a very long time to carry out some task, changing the BUFFERS command can only help if the disk is being accessed during the task. If the disk is never read or written to, the problem is not one of disk access but rather a problem of processor speed. Adjusting the BUFFERS command will not help with this problem.

If the DOS=HIGH command loads most of DOS into the HMA, then the BUFFERS will also be loaded into the HMA. In this case, increasing the number of buffers will not decrease memory available to applications. Loading DOS into the HMA is discussed in Chapter 4.

BUFFERS is a CONFIG.SYS command.

FASTOPEN

The FASTOPEN command increases the performance of your hard disk by keeping track of the names and locations of files and directories that you access. Without FASTOPEN, each time you access a file on a hard disk, DOS must search the directory structure to locate the starting position of the file, then retrieve the files chain of disk cluster addresses from the FAT, before it can actually access the file. FASTOPEN keeps this information in memory for recently opened files. When you then go back to those files, DOS does not have to search around various areas of the disk for this information. This can result in a significant improvement in the speed with which your computer retrieves information from the disk. FASTOPEN cannot be used on floppy drives.

When the FASTOPEN command is entered, you must specify the drive or drives for which it will track file information. Other drives are not affected. For each drive, you indicate how many files FASTOPEN should track. If a disk is partitioned into more than one logical drive, each drive that you want FASTOPEN to operate on must be specified. You may include up to 24 different drives in a FASTOPEN command. As a general rule of thumb, you should specify one file for every megabyte of hard disk space on a drive. The following command directs FASTOPEN to track 32 files for drive C: and eight files for drive D:

```
FASTOPEN C:=32 D:=8
```

This would be an appropriate setting for a computer that has a 40 meg hard drive positioned into a 32 meg C: drive and an 8 meg D: drive.

If you have expanded memory on your computer, you can use it to run FASTOPEN, thereby saving the conventional memory that FASTOPEN would otherwise use. To run FASTOPEN in expanded memory, the /X switch is used:

```
FASTOPEN C:=32 D:=8 /X
```

Expanded memory is discussed in Chapter 5.

WARNING

FASTOPEN can only be run once per session. If you want to change the FASTOPEN settings, you must restart your computer. If you are using the DOS 5.0 Shell, you must NOT run FASTOPEN from within the Shell. This can cause the computer to lock up, forcing you to reboot. This can result in loss of data and/or corrupted files on a disk. FASTOPEN should only be run from the command prompt, without having the Shell loaded. You should not temporarily suspend the Shell to go the command prompt for running FASTOPEN. If FASTOPEN was not run before starting the Shell, you should exit the Shell entirely to run it.

The FASTOPEN command can be included in your AUTOEXEC.BAT file, so that it loads automatically when you start your computer. It can also be placed in the CONFIG.SYS file, using the INSTALL command.

FASTOPEN is most effective with programs that open and close a lot of files, such as database management systems and computer language compilers. Other types of software may benefit more from the BUFFERS command or the SMARTDRIVE program. The best way to determine whether FASTOPEN is useful is to experiment with it. Try loading it, then run the programs you commonly work with. If you notice a speed increase, FASTOPEN is helping out. If you do not, your programs probably do not make use of the disk in ways that FASTOPEN can help with, so it should be removed to save the memory it occupies.

If you have a 386 or later computer with one Megabyte or more of memory, FASTOPEN can be placed in an Upper Memory Block, using the LOADHIGH command, as described in Chapter 5. This eliminates the loss of memory for conventional programs that would otherwise result from using the FASTOPEN program.

FASTOPEN is an external command.

SMARTDRIVE

The SMARTDRIVE program is what is known as a "disk-caching" program. Disk caching programs are used to speed up the performance of hard drives, and they often show dramatic improvement in the speed at which certain programs operate. These programs are termed disk-intensive because they perform a lot of disk accesses. Since disk accesses are by far the slowest operation the computer performs, speeding them up can be very beneficial.

A disk cache is like a huge disk buffer, as defined earlier by the BUFFERS command. When a disk cache is set up, the first time a sector is read from the disk, it is stored in the cache. Since the cache is much larger than buffers defined by the BUFFERS command, much more information is stored, thus increasing the likelihood that information needed from the disk will be found in memory.

SMARTDRIVE requires either extended or expanded memory. If you have both, extended memory is preferred. However, if you have significantly more expanded memory than extended, then you may want to use expanded for SMARTDRIVE, in order to conserve the extended memory for other purposes. SMARTDRIVE is actually a device driver, so it is loaded from the CONFIG.SYS file when the computer is booted. The following command in CONFIG.SYS loads SMARTDRIVE and specifies a 128K cache:

```
DEVICE=C:\DOS\SMARTDRV.SYS 128
```

Look

Notice that the complete path is required for the location of the SMARTDRV.SYS file. This is because it passed as a parameter to the DEVICE command, rather than being a command in its own right. When files are referenced as parameters, DOS does not search the current path for them, so you must specify the complete path to the file.

The number that specifies the size of the cache represents kilobytes; it may be anywhere from 128 to 8192. If the number specified is not a multiple of 16, SMARTDRIVE rounds it to the nearest multiple of 16. It is a good idea to use numbers that are multiples of 16, so that it is obvious exactly how big the cache will be. If no number is specified, SMARTDRIVE uses 256 as a default.

Normally, SMARTDRIVE stores the cache in extended memory. If you want SMARTDRIVE to use expanded memory for the cache, the /A switch is used. The following command creates a 1 megabyte cache in expanded memory:

```
DEVICE=C:\DOS\SMARTDRV.SYS 1024 /A
```

If you have a 386 or later CPU with extended memory, it is recommended that you not use the EMM386.EXE device driver to set up expanded memory for running SMARTDRIVE. Although this would work, EMM386.EXE is simply simulating expanded memory with some of the extended memory in the computer. SMARTDRIVE will perform better if you run it directly out of extended memory.

In addition to the size of the cache, there is a second parameter that may be passed to SMARTDRIVE, if you are running Windows 3.0 or later, or another program designed to adjust the cache size. These programs have the ability to dynamically change the size of the cache, depending on how much extended memory is needed by the programs they are running. If there's a lot of extended memory available, these programs create a large cache, but when a task is started that needs more extended memory, these programs shrink the cache down, thereby freeing up extended memory for the task. The second SMARTDRIVE parameter, then, represents the minimum cache size that will be provided, regardless of what a task requests. The following command sets up an initial cache of 1024K (which is 1 Megabyte), and specifies that the cache should never shrink below 256K:

```
DEVICE=C:\DOS\SMARTDRV.SYS 1024 256
```

SMARTDRIVE takes up 15K of conventional memory, in addition to the extended or expanded memory used by the cache itself. It may take even more conventional memory than this, depending on the size of the cache and the size of the largest track on your hard disk. You can see exactly how much conventional memory SMARTDRIVE uses by running the MEM/C command at the DOS prompt.

15K is not an enormous amount of memory, but in some cases it is enough to make the difference between whether a program will run or not. Consequently, in some cases you may be better off not using SMARTDRIVE, to leave all of the conventional memory for programs. SMARTDRIVE cannot be loaded into a UMB with the LOADHIGH command, so if you use SMARTDRIVE, the 15K will have to come out of conventional memory. Attempting to load SMARTDRIVE into a UMB may prevent it from functioning properly. UMBs and the LOADHIGH command are discussed in Chapter 4.

SMARTDRV.SYS is a program that is provided with both Microsoft Windows and DOS 5.0. You should use the most recent version you have, unless specifically directed otherwise by the Microsoft documentation. The version of SMARTDRIVE that comes with DOS 5.0 is more recent than the version that shipped with Windows 3.0, and should therefore replace the Windows 3.0 version.

In general, you want to use SMARTDRIVE only if you have sufficient memory resources to properly support it. If your system has at least 512K of extended memory, or at least 256K of expanded memory, then it is probably a good use of these resources to install SMARTDRIVE. You should leave sufficient extended or expanded memory for whatever other purposes you want to use them for, then provide the largest possible cache, up to 2 megabytes. To determine how large a cache your system can handle, install the largest possible cache and test your programs. If they don't run, reduce the size of the cache a little and try again. When all of your programs run successfully, you have found the largest cache you can have for that set of programs.

It is a good idea to use a disk compaction program to keep your disk well organized, whether or not you are using SMARTDRIVE. If you are using SMARTDRIVE, the disk compaction program will keep SMARTDRIVE performing as efficiently as possible, because it keeps related information in the same area of the disk.

SMARTDRV.SYS is a CONFIG.SYS device driver.

RAMDRIVE — Speeds Up Your System

RAMDRIVE is a program that allows you to create a "RAM disk." A RAM disk is an area of memory that is set up to simulate a disk drive. Once a RAM disk is set up, you can reference it just as you would any other drive. However, because it consists of electronic memory instead of a mechanical drive, a RAM disk operates at many times the speed of even a very fast hard disk.

RAMDRIVE is not generally as useful as SMARTDRIVE, because a RAM disk has to have specific files copied to it, and it is then only useful for speeding up access to those files. SMARTDRIVE, on the other hand, speeds up access to the hard drive, regardless of which files are being accessed.

However, if you have a program that frequently accesses one or more fairly small files, you may achieve a greater performance increase by using RAMDRIVE than SMARTDRIVE. Also, if your computer does not have a hard drive, then SMARTDRIVE cannot be used, and your disk accesses will be very slow since they are referencing floppy drives. In this case, RAMDRIVE can make an enormous difference in your computer's performance.

Since RAMDRIVE creates a "drive" in memory, the drive is volatile — if the computer is turned off, the contents of the drive are lost. Consequently, RAM disks should never be used to store data files. They should be used for files that are referenced, such as spell checking dictionaries in word processors. If you want to use a RAM disk to speed up access to a data file for an operation like spell checking, the file should be copied to the RAM disk before the operation, then, if it was changed during the operation, back to the real disk as soon as the spell checking is completed.

One of the most useful ways to use a RAM disk is with programs that create temporary files in a directory specified by an environment variable. An environment variable is a name that resides in the DOS environment, and that contains some value. Programs can reference environment variables. Many programs will look for an environment variable named TEMP that will identify the directory in which they should store their temporary files. You can define this variable to contain the C:\TEMP directory by entering the following command:

```
SET TEMP=C:\TEMP
```

If you are using a RAM drive, then you can store the name of a directory on the RAM drive in the TEMP environment variable. Now these programs will store their temporary files on the RAM drive, which can greatly increase the speed of the programs.

RAMDRIVE is actually a device driver, so it is loaded from the CONFIG.SYS file when the computers is booted. The following command in CONFIG.SYS loads RAMDRIVE and specifies a 200K RAM disk in conventional memory:

```
DEVICE=C:\DOS\RAMDRIVE.SYS 200
```

The size parameter, 200 in the above command, indicates the capacity of the RAM disk in kilobytes. You may specify any number from 16 through 4096. The default value is 64. When the above command executes, the following messages appear on the screen:

```
Microsoft RAMDrive version 3.06 virtual disk D:
   Disk size: 200K
   Sector size: 512 bytes
   Allocation unit: 1 sector
   Directory entries: 64
```

When RAMDRIVE creates a RAM disk, it determines what is the last drive letter in use by your computer, and assigns the next drive letter to the RAM disk. Thus, the last drive letter on the computer that produced the above example was C: and RAMDRIVE therefore assigned the letter D: to the RAM disk

RAM disks are set up to have 512 byte sectors, just like all physical disks that are found on PCs. You may optionally specify an alternative sector size for the RAM disk, though there is rarely any reason to do this and it is strongly recommended that you do not. This is done by specifying the sector size immediately after the disk capacity. Valid sector sizes are 128, 256, and 512 bytes. The following command specifies a 200K RAM disk with 256 byte sectors:

```
DEVICE=C:\DOS\RAMDRIVE.SYS 200 256
```

Just as physical disks have a specified number of entries that can be stored in the root directory, so do RAM disks. The default number of root directory entries for a RAM disk is 64. You may change this value by appending the maximum number of root directory entries to the end of the command. If you specify the number of root directory entries, you must also specify the sector size. The following command sets up a 200K RAM drive that has 512 byte sectors and a root directory that can store up to 100 entries:

```
DEVICE=C:\DOS\RAMDRIVE.SYS 200 512 100
```

You may specify anywhere from 2 to 1024 root directory entries. Increasing the size of the root directory increases the total amount of memory used for the RAM drive. If there is not enough memory to set up the RAM drive as you have specified, RAMDRIVE will try to set it up with a root directory that can contain 16 entries.

By default, RAMDRIVE creates the RAM disk in conventional memory. This is the worst place to put it, since conventional memory is needed for running programs. However, it is also the only type of memory that every computer is guaranteed to have, so it makes sense to default to this and then let you change it. To specify that the RAM disk should go into expanded memory, the /E switch is used. If you want to put the RAM disk into extended memory, the /A switch is used.

You may create as many RAM disks as you like, by specifying multiple `DEVICE=RAMDRIVE.SYS` lines in the CONFIG.SYS file.

In most cases, setting up a RAM disk in conventional memory is not recommended. The performance gains are generally not worth the lost memory. However, if your computer has expanded or extended memory, a RAM disk may be worth using.

If you have a 386 or later CPU with extended memory, it is recommended that you not use the EMM386.EXE device driver to set up expanded memory for running RAMDRIVE. Although this would work, EMM386.EXE is simply simulating expanded memory with some of the extended memory in the computer. RAMDRIVE will perform better if you run it directly out of extended memory.

Disk Fragmentation

Disk fragmentation is the most common cause of deteriorating disk performance, and is also one of the simplest problems to overcome. Disk fragmentation occurs as files are added, deleted, and expanded on a disk. For example, consider a hard disk that has four sectors per cluster. Disk space is allocated in 2048-byte chunks. Say you create a 2000-byte file that is stored in cluster 5. Now you create another 2000-byte file that is stored in cluster 6. So far so good, but what happens when you want to add more information to the first file? Since cluster 6 is in use, cluster 7 gets allocated to the first file. As some files grow, others shrink, and yet others are deleted, the pattern of available clusters on the disk becomes more and more fragmented.

On a floppy disk, this problem is easy to fix: simply use the COPY or XCOPY command to copy all of the files to another disk. These commands copy the files into contiguous sectors on the new disk, thereby eliminating fragmentation. If you have two hard disks available, and there is enough free room on one to duplicate the other, this technique can be used there as well. But this is rarely the case, and even when a second hard disk is available, this is a clumsy approach. For this reason, a variety of software developers have created programs known as "disk compression," "disk compaction," or "disk optimizer" utilities. These programs can unfragment a disk without copying everything to a second disk (they are available from software dealers). If you purchase one, make sure it can handle interruptions without damaging the disk or losing data, since there is no guarantee that a power failure or some other problem won't arise while the program is running. It is also a good idea to back the hard disk up onto floppies before using one of these programs, until you are confident that it is working properly and that your files are not being damaged in the process.

Before running a disk compression utility, you should delete unnecessary files. Deleting them after the compression is done will simply fragment the disk again. It is also a good idea to run the CHKDSK command, discussed below, to recover any lost clusters.

Adjusting the Interleave

DOS 5.0 does not provide any programs or commands for changing the interleave of a disk. However, adjusting the interleave to be appropriate for your PC can make a significant difference in the performance of your hard disk. There are programs available from software dealers that can be used to adjust the interleave. These programs are usually capable of analyzing the current interleave as well, and determining what interleave will work best on your computer.

Chapter 7

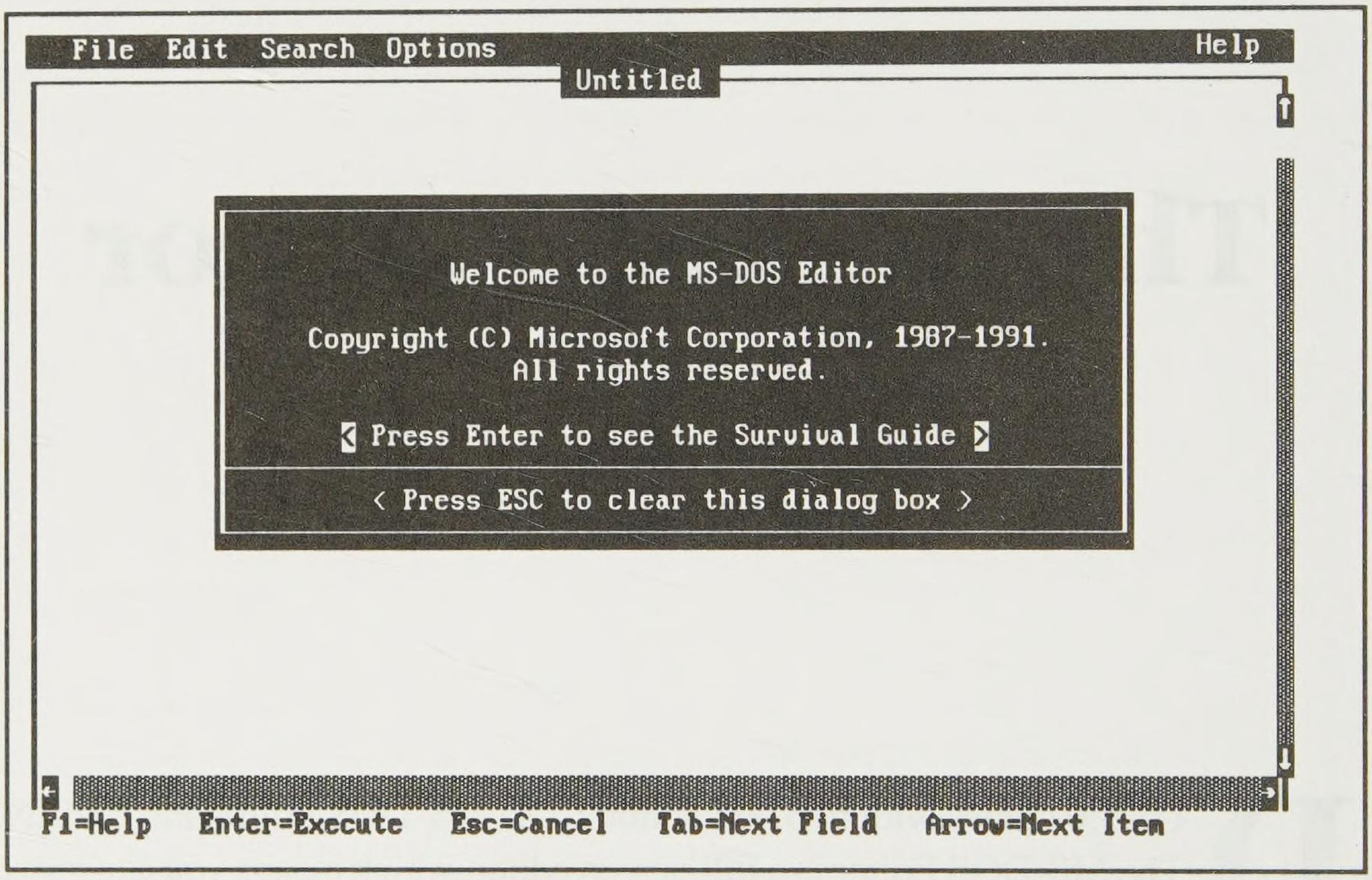
The MS-DOS Editor

DOS 5.0 comes with two text editors: The MS-DOS Editor and EDLIN. The MS-DOS Editor is a full-screen editor, which means you can move the cursor around the screen and scroll through files, as you can with word processors. MS-DOS Editor is actually a mini word processor. EDLIN, on the other hand, is a line editor, which means that you work on one line of a file at a time. EDLIN is a hold over from previous versions of DOS. Because the MS-DOS Editor is so much better, there is no longer any reason to use EDLIN.

Both of these editors produce plain ASCII text files, which means they have only the standard ASCII characters in them. A complete table of the ASCII characters and their codes appears in the appendices. Batch files, the CONFIG.SYS file, the text files that the DOS 5.0 Shell uses, and other files used by DOS, all must contain standard ASCII codes only. Most word processors do not produce plain ASCII files unless you specifically tell them to.

Starting and Leaving the MS-DOS Editor

You start the MS-DOS Editor with the EDIT command. You may include a file name as a parameter on the command line, in which case the specified file is loaded when the editor starts. If no file name is specified, the MS-DOS Editor Welcome Screen shown in Figure 7-1 will be displayed.



Welcome to the MS-DOS Editor

Copyright (C) Microsoft Corporation, 1987-1991.
All rights reserved.

< Press Enter to see the Survival Guide >

< Press ESC to clear this dialog box >

Figure 7-1 — Welcome Screen for MS-DOS Editor.

When the screen in Figure 7-1 is displayed, pressing Enter← takes you to the "Survival Guide," which is an extensive system of online help for using the various features of MS-DOS Editor. Pressing esc leaves you with a blank editing screen. This is the same screen shown in Figure 7-1, except the Welcome dialog box is missing. From the blank editing screen you can start creating a file, or bring in a new file to edit by using the File menu.

When there is no file loaded into the editor, the word "Untitled" appears above the editing area. When a file is loaded, the name of the file appears here.

Before going any further, it is useful to know how to exit the Editor. Across the top of the editor screen are the words File, Edit, Search, and Options. Each of these represents a menu that has various items on it for using the Editor. To display one of these menus, hold down alt and press the first letter of the menu you want. The File menu has the Exit option on it, so to display the File menu, hold down alt and press F. (If one of the other menus is already displayed, use ← to move to the File menu.) The letter X of the Exit option is reverse-highlighted, indicating that X is a shortcut for selecting Exit. Press X. If you have changed anything on the screen since starting the editor, you will be prompted as to whether or not you want to save your changes. After you answer, you will be returned to where you invoked the editor — either the command line or the Shell. If no changes were made, pressing X will exit the Editor.

Navigating in the Editor

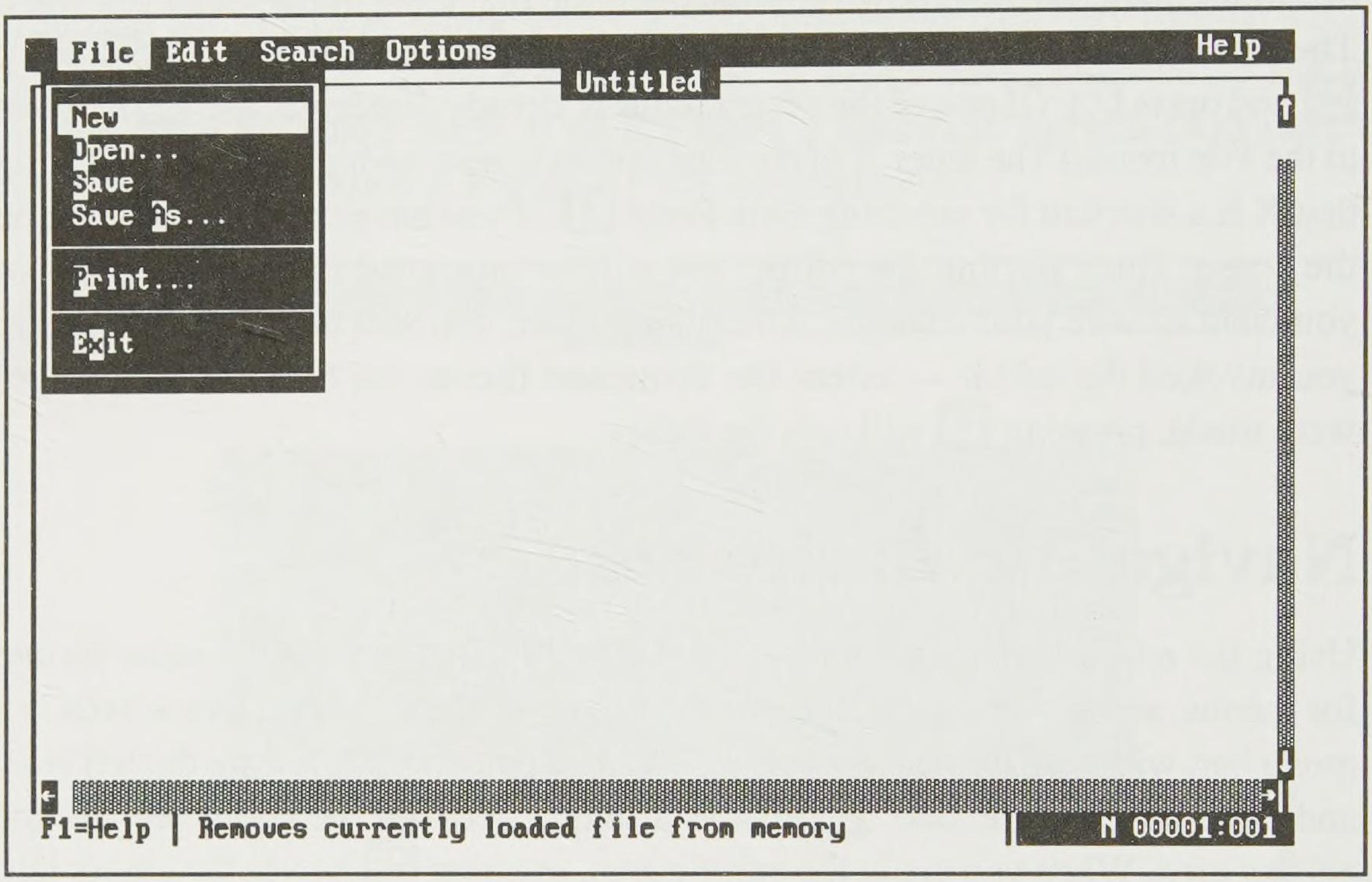
Using the editor is quite similar to using the Shell — both use the same format for menus, scroll bars, and dialog boxes. The upper line of the Editor screen is a menu bar, with five options on it: File, Edit, Search, and Options on the left side, and Help on the right side. alt acts as a toggle between the menu bar and the editing area. When you are in the editing area, pressing alt moves the cursor into the menu bar with the File option highlighted. When an item on a menu is highlighted, pressing alt returns you to the editing screen.

When you have highlighted one of the options on the menu bar, you can press Enter↵ to pull down a menu for that option. Alternatively, you go from the editing screen directly to a pull-down menu by holding down alt and pressing the first letter of the option name. For example, to go from the editing screen to the Edit pull-down menu, hold down alt and press E.

Once you have entered the menu bar, you may use ← and → to move from one menu to the next. Pressing Enter↵ causes the highlighted menu to display. The same techniques are used to select items on the menus: each item will have one letter of its name reverse highlighted. Pressing that letter automatically executes the corresponding operation. Alternatively, you may use the arrow keys to highlight an option, then press Enter↵ to execute it.

The File Menu

Files are selected, saved, and printed, using the File menu shown in Figure 7-2.

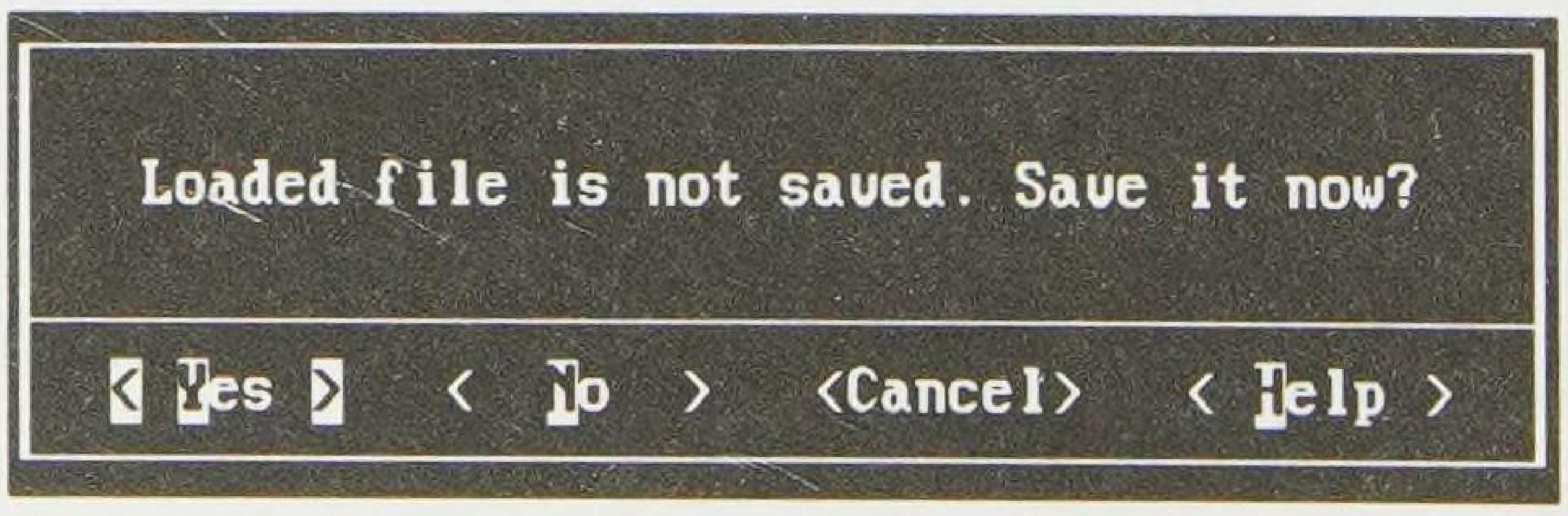


F1=Help | Removes currently loaded file from memory

N 00001:001

Figure 7-2 — File Menu of MS-DOS Editor.

If you started the editor with no file name on the command line, then the word Untitled appears at the top of the editing screen, and there is no file loaded into the editor. You can go ahead and start typing, and when you are finished, save what you have typed in a file. If you specified a file name when you started the editor, then its name appears at the top of the screen and you can go ahead and edit it. To create a new file, select New from the File menu. If you have changed the file that is currently displayed, you will be prompted with the dialog box shown in Figure 7-3.



Loaded file is not saved. Save it now?

Figure 7-3 — Save File Dialog Box.

Selecting Yes or pressing Enter←┘ saves the changes you have made, while selecting No cancels the changes. Either of these options then leads to a blank editing screen, with the word Untitled at the top. You may go ahead and create a new file on this screen, and when you are done, save it using the Save option on the File menu.

Sometimes you want to edit an existing file, but then save it under a new file name, so that the original file is left unchanged. This can be done with the Save As option on the File menu, which prompts you with the dialog box shown in Figure 7-4.

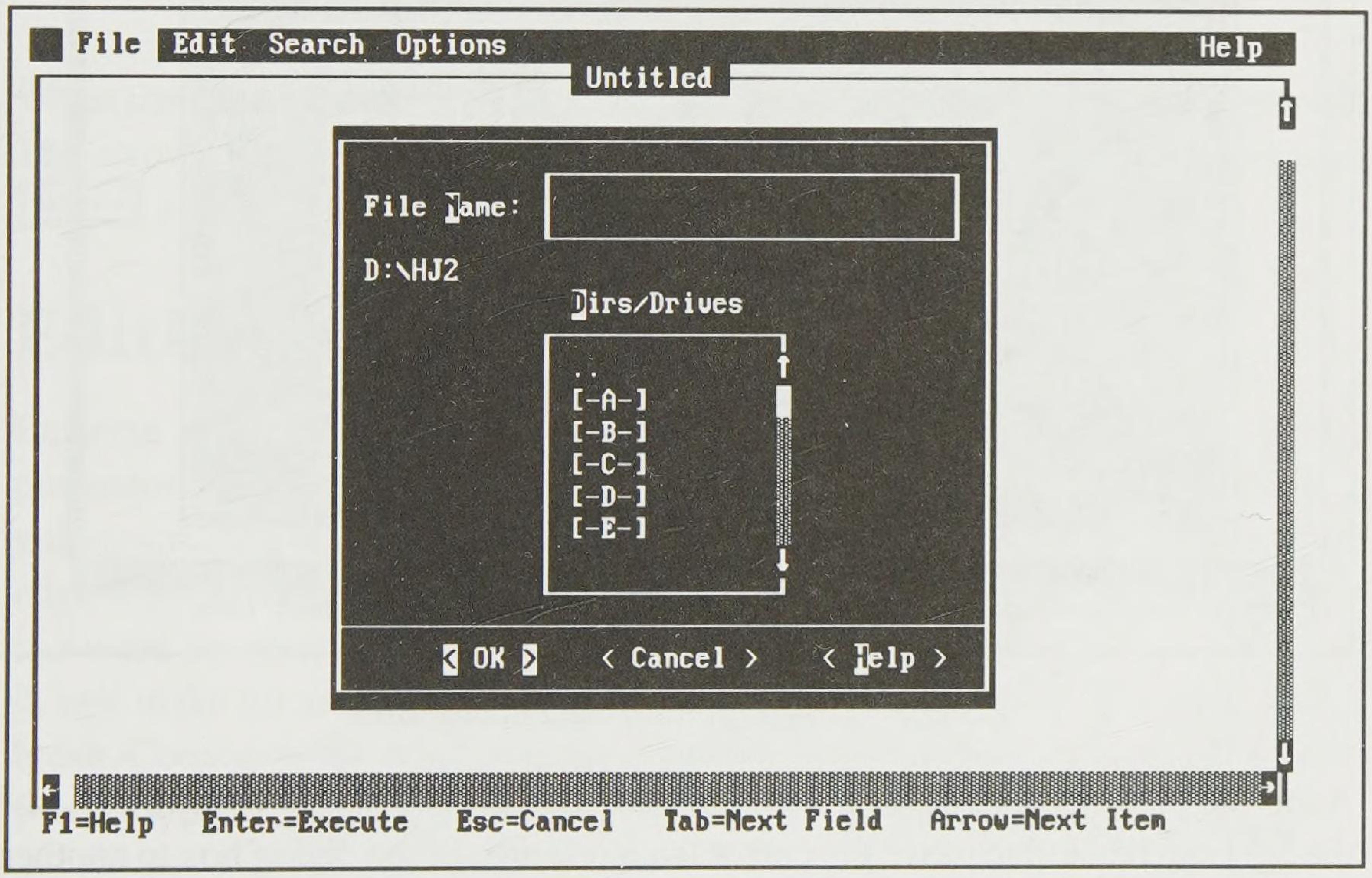
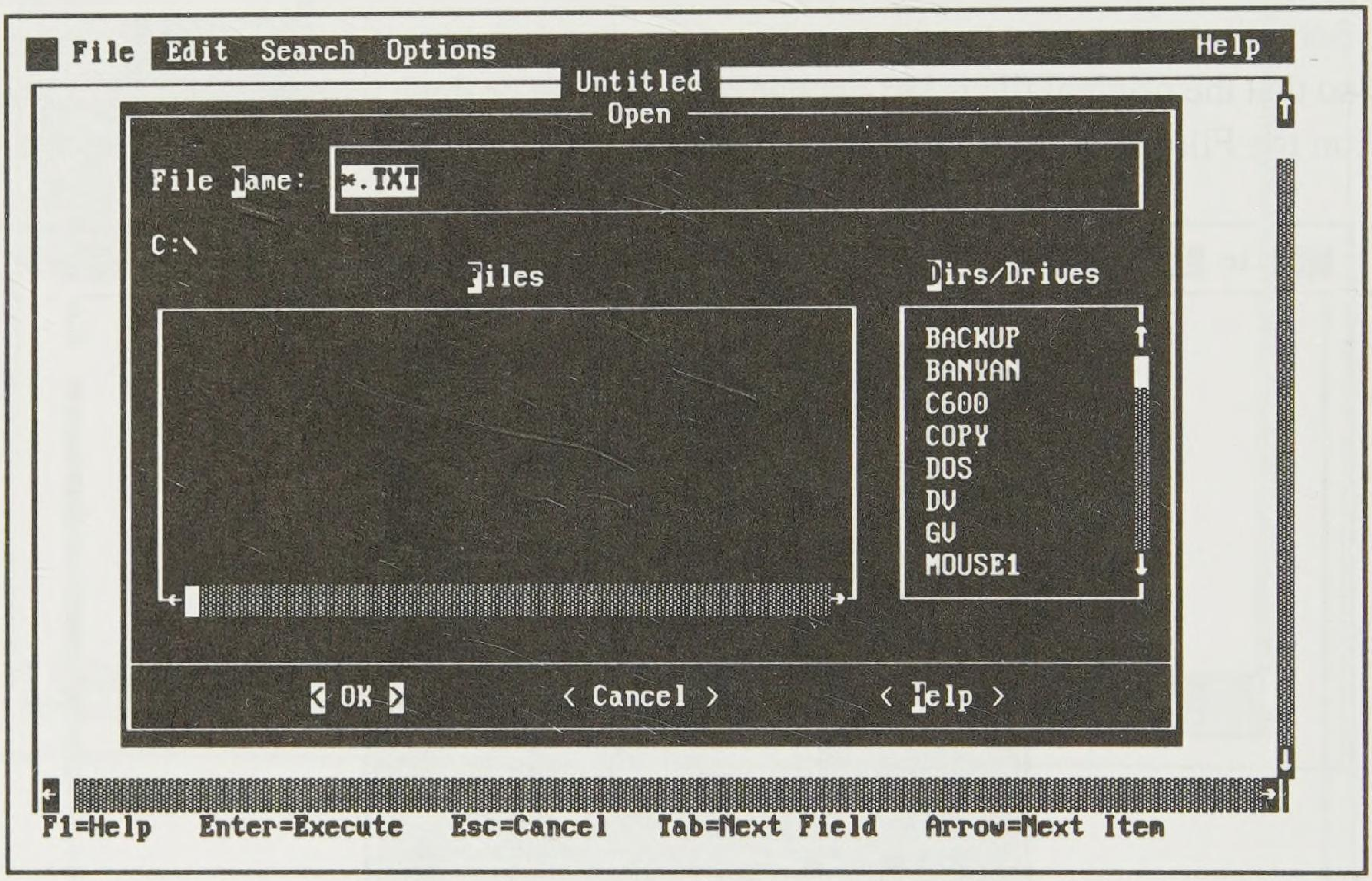

File Edit Search Options Help
Untitled
File Name:
D:\HJ2
Dirs/Drives
..
[-A-]
[-B-]
[-C-]
[-D-]
[-E-]
< OK > < Cancel > < Help >
F1=Help Enter=Execute Esc=Cancel Tab=Next Field Arrow=Next Item

Figure 7-4 — Save As Dialog Box.

When the Save As dialog box opens, the cursor will be in the File Name text box. Typing a file name here and pressing Enter←┘ will save the file under that name, in the current directory. If you want to save it in a different directory, or even in a different directory on a different drive, you can use the tab to move the cursor into the "Dirs/Drives" list. You may scroll through this list, and as the cursor moves from one item to the next, the selected drive or directory appears in the File Name text box. Selecting a new drive or directory causes the "Dirs/Drives" list to change, to reflect the new directory or drive.

When you have found the directory in which you want to save the file, you can type its name into the File Name text box and press Enter.

To retrieve an existing file, select Open from the File menu. This will bring up the Open dialog box shown in Figure 7-5.


File
Edit
Search
Options
Help
Untitled
Open
Files
Dirs/Drives
BACKUP
BANYAN
C600
COPY
DOS
DV
GV
MOUSE1
< OK >
< Cancel >
< Help >
F1=Help
Enter=Execute
Esc=Cancel
Tab=Next Field
Arrow=Next Item

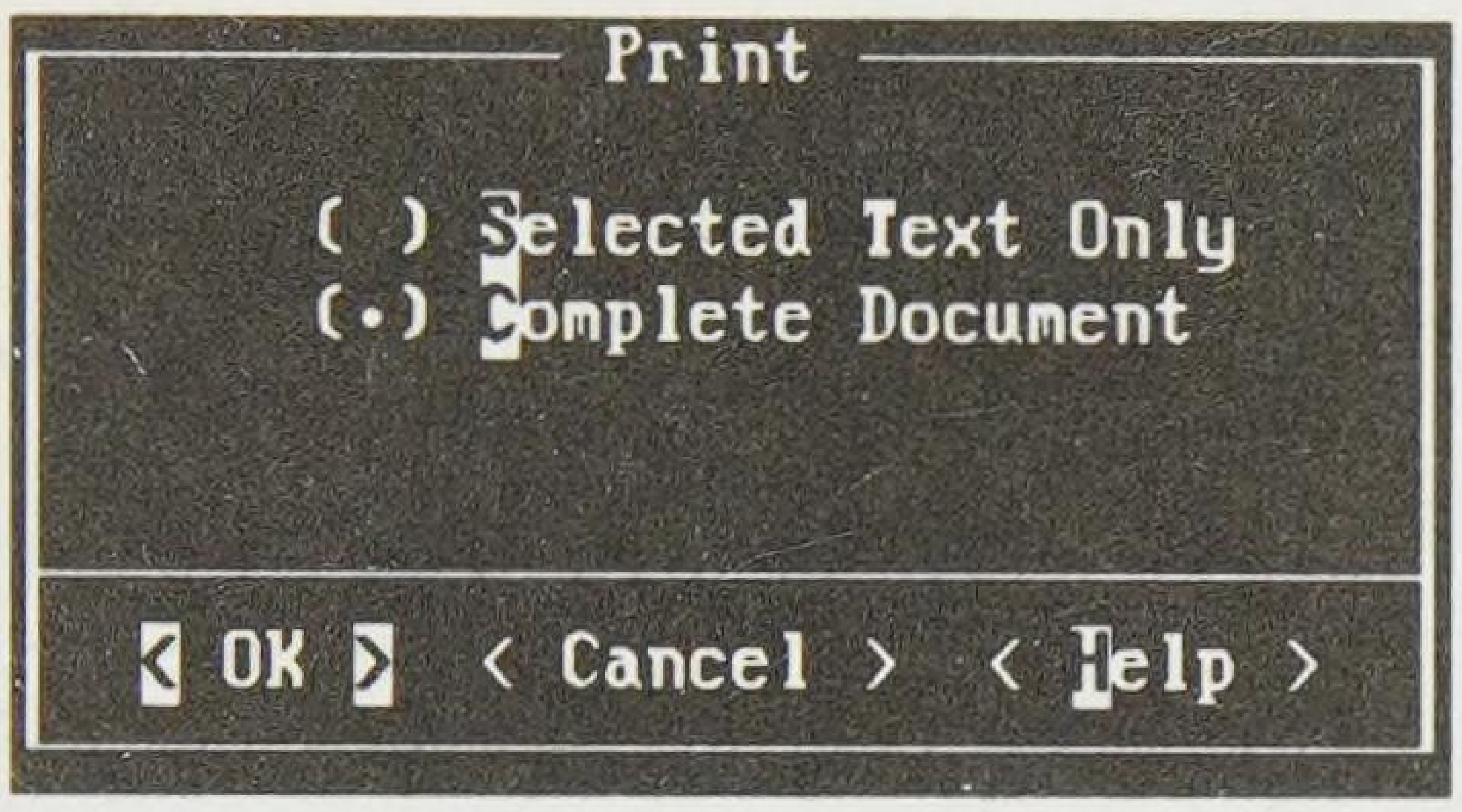
File Name: *.TXT

C:\

Figure 7-5 — Open A File Dialog Box.

As in the previous example, the cursor starts out in the File Name text box, and the tab can be used to move the cursor from one area of the dialog box to another. You may use wild cards to change the files that appear in the Files list. To select one of these files, move the cursor into that area, and use the arrow keys to move around in the list of files. The currently highlighted file name will appear in the File Name text box, and pressing Enter loads the file into the editor. Alternatively, you may simply type the file name into the File Name text box and press Enter. Again, you may use the "Dirs/Drives" list to change the directory and/or drive that is selected.

You may print a file that is loaded into the editor by selecting Print from the File pull-down menu. This brings up the dialog box shown in Figure 7-6.



( ) Selected Text Only
(•) Complete Document

Figure 7-6 — File Menu Print Option Dialog Box.

When the Print dialog box appears, the "Complete Document" option is selected. The arrow keys can be used to select the "Selected Text Only" option. Pressing Enter↵ accepts the selected option and prints the corresponding text.

Editing Files

Entering text into a new file is simple: just type away. However, unlike word processors that automatically wrap at the end of a line, MS-DOS Editor keeps on going, scrolling text to the left to make room for what you are writing at the right edge of the screen. This is in keeping with the purpose of a text editor, as opposed to a word processor: a text editor is intended to produce text exactly as you write it, and make no assumptions about how it should appear or where lines should break. Consequently, when you reach a point where you want a new line to start, you must press Enter↵, just as you would on a typewriter.

Moving Around in a File

There are several ways of moving the cursor around in a file you are editing. In addition, other keys are used for deleting text at and around the cursor, and for performing search and replace operations for text. These keys are all summarized in Table 7-1.

Table 7-1 — Editing Keys for the MS-DOS Editor.

Key	Effect
←BkSp or Ctrl-H	Deletes character to left of cursor
delete or Ctrl-G	Deletes character at the cursor, or marked block
Ctrl-T	Deletes from cursor to end of word
Ctrl-Y	Deletes line cursor is on
Ins or Ctrl-V	Toggles between insert and overstrike mode
→	Moves cursor one space to the right
←	Moves cursor one space to the left
↑	Moves cursor up one line
↓	Moves cursor down one line
home	Moves cursor to beginning of line
end	Moves cursor to end of line
Ctrl-←	Moves cursor one word to the left
Ctrl-→	Moves cursor one word to the right
Ctrl-Enter↵	Moves cursor to beginning of next line
Ctrl-Q-E	Moves the cursor to the top of the screen
Ctrl-Q-X	Moves the cursor to the bottom of the screen
Ctrl-Q-R	Moves the cursor to the beginning of the file
Ctrl-Q-C	Moves the cursor to the end of the file

continued...

...from previous page (Table 7-1)

Key	Effect
Ctrl-↑ or Ctrl-W	Scrolls up one line; cursor does not move
Ctrl-↓ or Ctrl-Z	Scrolls down one line; cursor does not move
PgUp or Ctrl-R	Scrolls up one screen
PgDn or Ctrl-C	Scrolls down one screen
Ctrl-PgUp	Scrolls left one screen
Ctrl-PgDn	Scrolls right one screen
Ctrl-Q-F	Searches for text
Ctrl-Q-A	Searches for and replace text
Ctrl-L	Repeats last search

The Edit Menu

When editing a file, you may mark a block of text that can then be manipulated. For example, you may want to move a sentence from one part of a file to another, or you may just want to delete it. To mark a block of text, put the cursor at the beginning of the block, then hold down Shift while you move the cursor to the end of the block. You may now use any of the options on the Edit menu to affect the block. The Edit menu is shown in Figure 7-7.

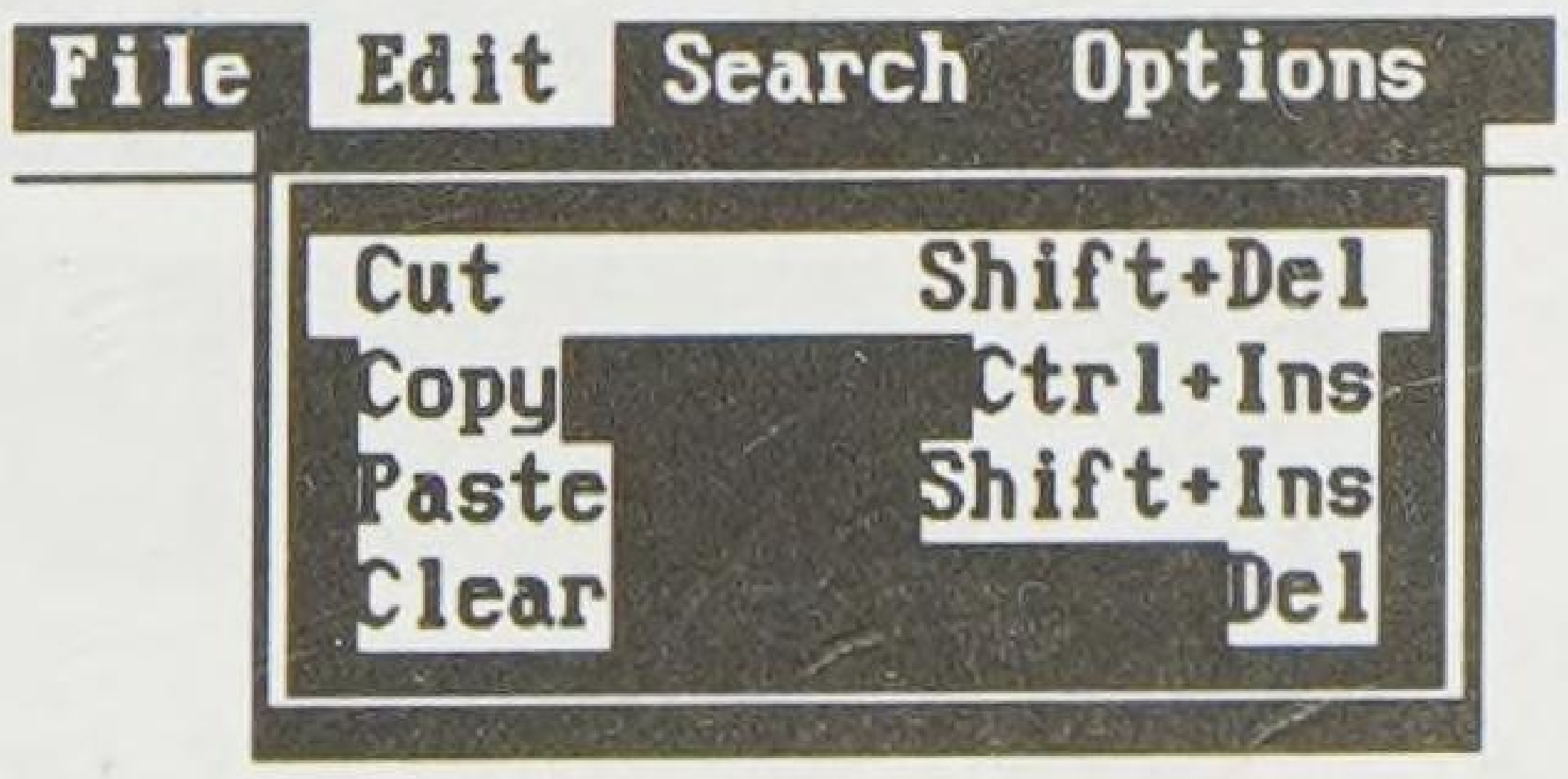


Figure 7-7 — Edit Menu of MS-DOS Editor.

The Edit menu also shows the keyboard shortcut for each option. The Cut and Copy options store the marked block in something called the "clipboard." This is a temporary holding area for text that has been cut or copied. The clipboard can only hold one block of text at a time, so it is important to be finished with its current contents before storing something else in it. When another block of text is stored in the clipboard, the first block is gone.

The contents of the clipboard may be pasted somewhere else in the file. If the Cut option was chosen, then the marked block is removed from the screen when the Cut takes place. Using the Paste option to place the text somewhere else has the effect of moving the text from one place to another. If the Copy option was chosen, then using the Paste option to place the text somewhere else has the effect of duplicating the block.

The Clear option, which is the same as pressing delete, erases the marked block from the screen, but does not store it in the clipboard. Consequently, the clipboard contents are not affected, and it is impossible to retrieve the deleted text. The Ctrl-T and other keyboard commands for deleting sections of text are similar to Clear — they do not store the deleted text in the clipboard, so they are irretrievable deletions.

The Search Menu

You may locate a text string in a file, and optionally change it to something else, using the Search menu, which is shown in Figure 7-8.

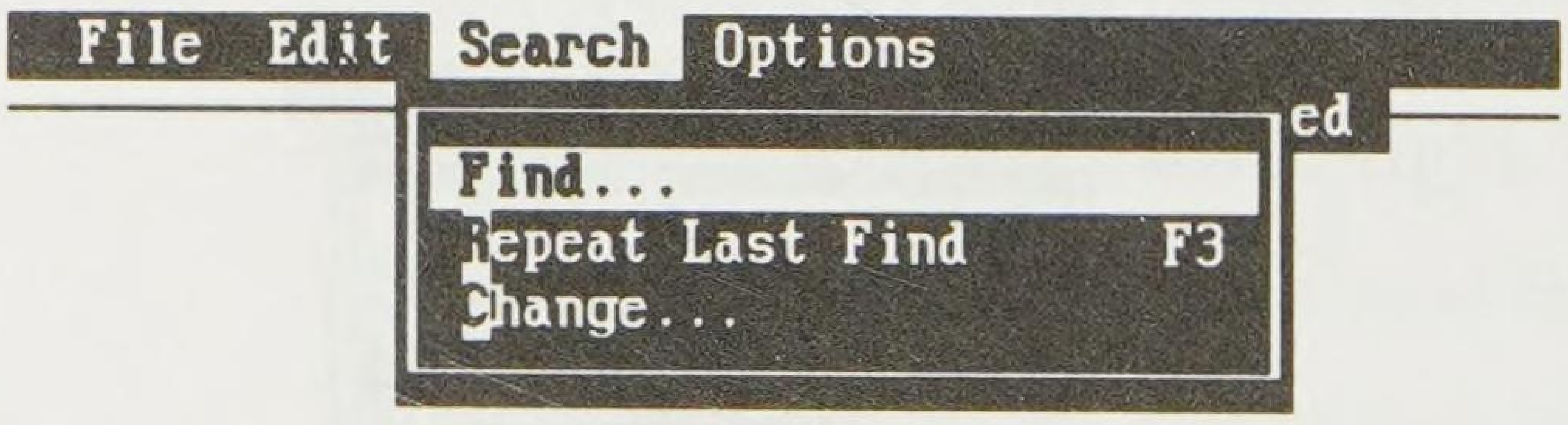
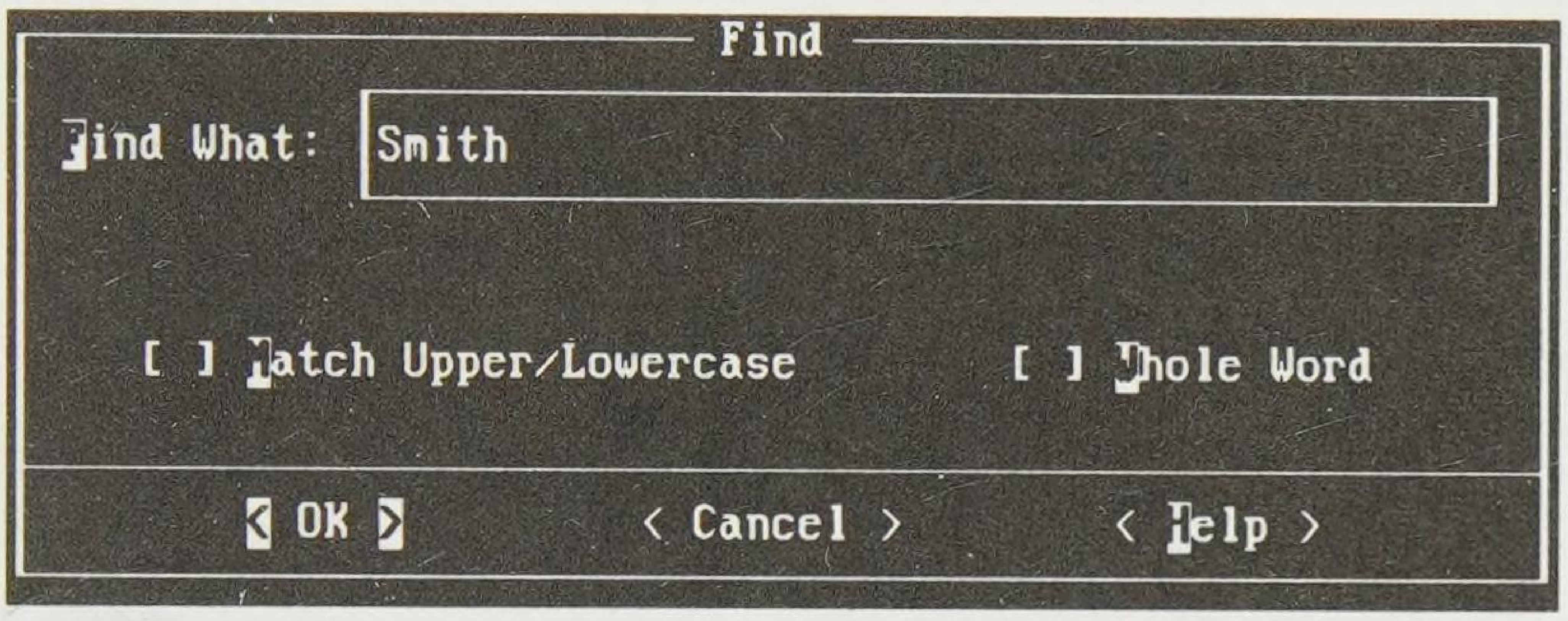


Figure 7-8 — Search Menu of MS-DOS Editor.

Selecting Find from the Search menu brings up the Find dialog box shown in Figure 7-9.



Find What: Smith

Figure 7-9 — Dialog Box for Finding Text.

SHORTCUT

The Ctrl-Q-F combination is a shortcut for bringing up the Find dialog box, though the shortcut isn't shown on the Search menu. The word that appears at the current cursor location is automatically placed in to the "Find What" text box. You may edit this to search for some other text, or accept it by pressing Enter.

The tab can be used to move the cursor into the "Match Upper/Lowercase" and "Whole Word" check boxes. Once the cursor is in one of these check boxes, the check mark can be turned on and off by pressing Shift. Match Upper/Lowercase makes the search case sensitive, and Whole Word specifies that only whole word matches should be identified. If Whole Word were checked, then the letters "FOR" in CALIFORNIA would not be selected, if the search was for the word FOR.

When the OK button is selected, or Enter is pressed, the cursor is moved to the first occurrence of the string being searched for. If the string is not found, a "Match not found" dialog box will be shown in the middle of the screen.

If the string is found, the cursor is positioned at the end of it. To search for the next occurrence of the string, all you have to do is press Ctrl-L, or F3. Alternatively, you can select Repeat Last Find from the Search menu.

SHORTCUT

To perform a Search and Replace operation, select Change from the Search menu. A shortcut is to press Ctrl-Q-A. This brings up the Change dialog box shown in Figure 7-10.

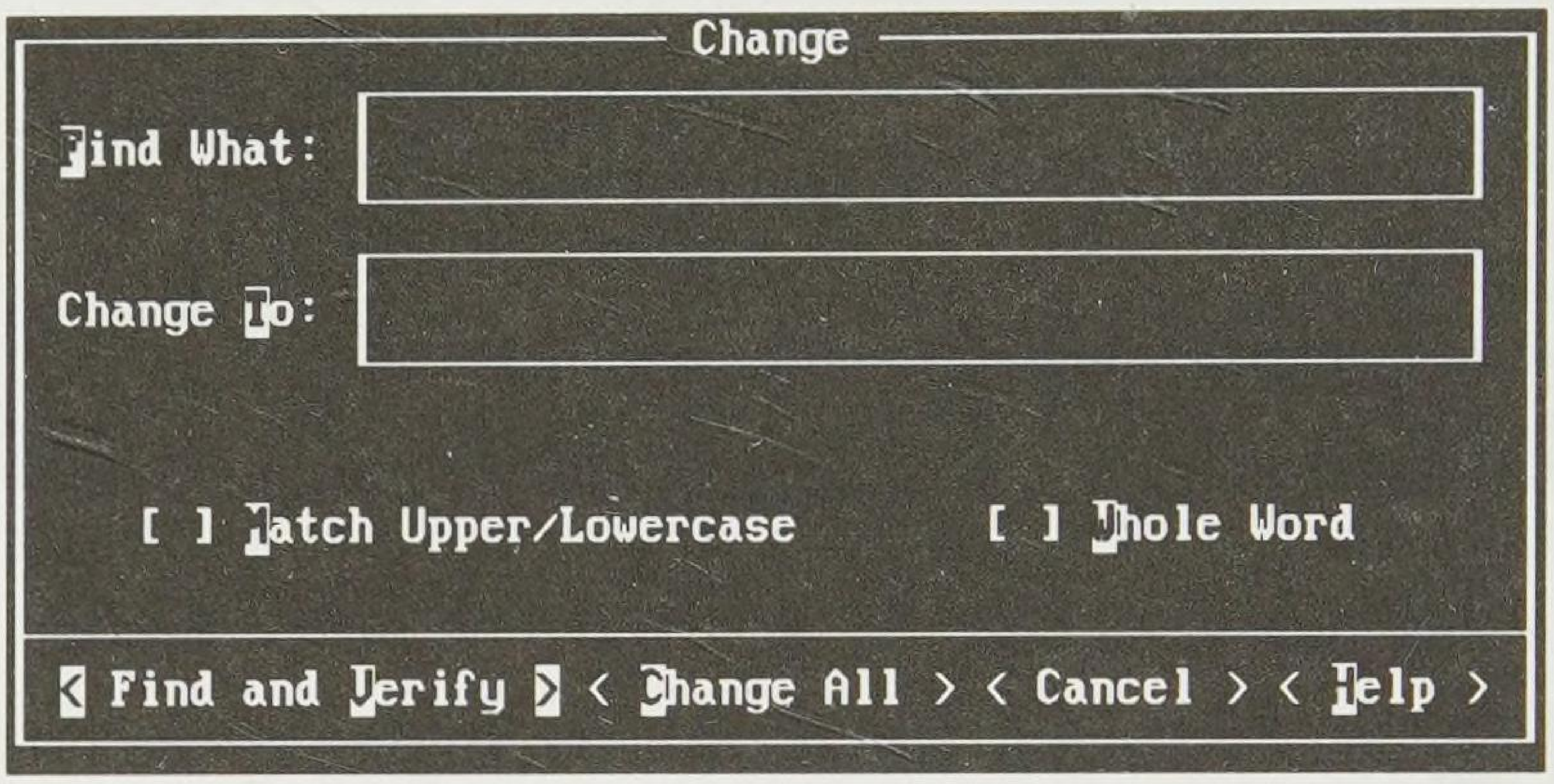

Change
Find What:
Change To:
[ ] Match Upper/Lowercase
[ ] Whole Word
< Find and Verify >
< Change All >
< Cancel >
< Help >

Figure 7-10 — Dialog Box for Search and Replace Operation.

This dialog box is similar to that shown in Figure 7-9, except that the "Change To" text box as been added. The tab is used to move the cursor into this text box, where the replacement text is typed. The "Find and Verify" option at the bottom of the dialog box tells the Editor to prompt you for confirmation before it changes each occurrence of the search string. The "Change All" option tells the Editor to change all occurrences, without prompting you for confirmation.

The Options Menu

The Options menu provides the ability to customize the colors and other aspects of the screen display, and to specify the location of the Editor help files. The Options menu is shown in Figure 7-11.

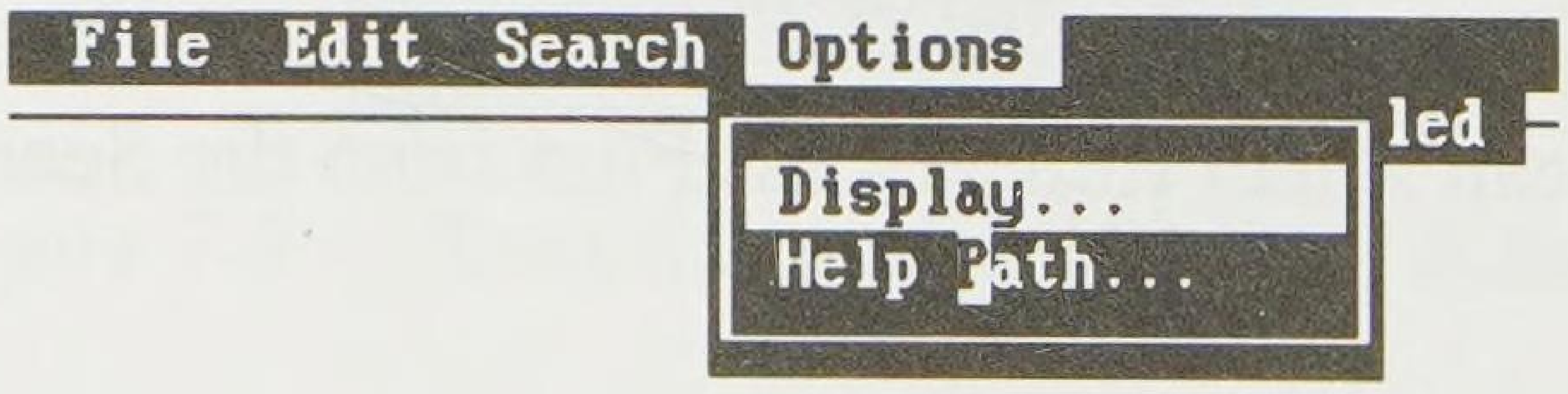
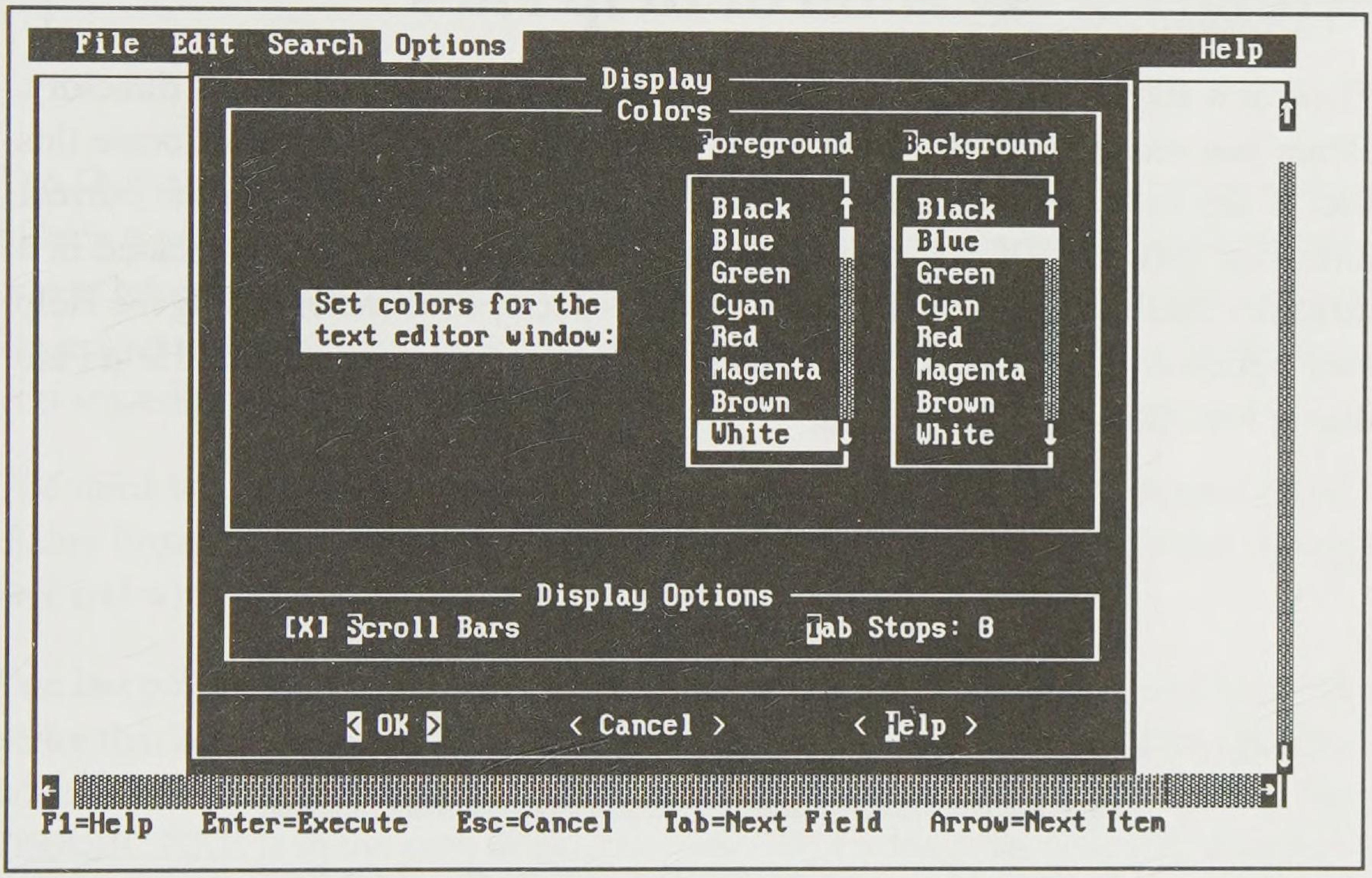

File
Edit
Search
Options
led
Display...
Help Path...

Figure 7-11 — Options Menu of MS-DOS Editor.

Adjusting the Colors and Screen Display

Selecting Display from the Options menu allows you to adjust the colors that appear on the screen, to specify whether or not the scroll bars should appear, and to set the number of spaces that should be displayed for a tab stop. The Display option brings up the Display dialog box shown in Figure 7-12.


File
Edit
Search
Options
Help
Display
Colors
Foreground
Background
Black
Blue
Green
Cyan
Red
Magenta
Brown
White
Black
Blue
Green
Cyan
Red
Magenta
Brown
White
Display Options
< OK >
< Cancel >
< Help >
F1=Help
Enter=Execute
Esc=Cancel
Tab=Next Field
Arrow=Next Item

Set colors for the text editor window:

[X] Scroll Bars

Tab Stops: 8

Figure 7-12 — Dialog Box for Adjusting Screen Display.

As you select different colors for the foreground and background, the text "Set colors for the text editor window:" changes to reflect the selected colors. This allows you to preview the effect of different settings, without having to commit to them.

The Scroll Bars check box can be used to turn off the scroll bars. Turning off the scroll bars provides a little more room on the screen to display your file. If you do not use a mouse with the editor, then there is little reason to have the scroll bars on.

LoOk

The Tab Stops setting is the number of spaces that tabs will be expanded to. Note that this affects the way the file appears on the screen, but it does not affect the file itself in any way. A tab stop symbol is the same symbol, regardless of this setting. Eight spaces is a fairly standard setting, though for some purposes, such as editing computer programming language files, two spaces is also common.

Setting the Location of Help Files

There is a file named EDIT.HLP that is normally stored in the DOS directory. When you request a help screen in the MS-DOS Editor, it needs to locate this file. If the file is in a directory specified in the path, or if it is in the current directory, then MS-DOS Editor can find it. However, if the file is located in a directory that is not in the path, then you may specify its directory using the Help Path option on the Options menu. The Help Path option brings up the Help Path dialog box shown in Figure 7-13.

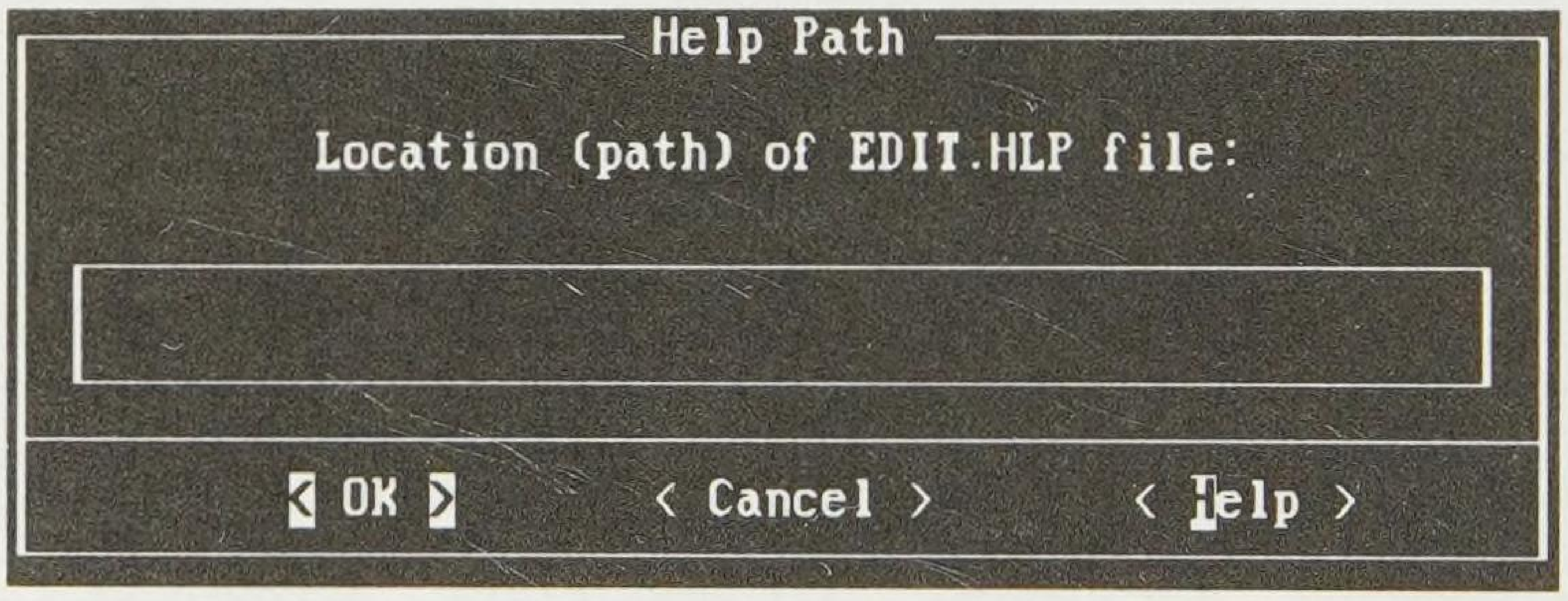


Figure 7-13 — Dialog Box for Setting the Help Path.

Using Online Help in the Editor

The MS-DOS Editor contains extensive online help for learning about the editor and its features. There are several ways to access this help system. Pressing F1 at any time brings up context-sensitive help. For example, pressing F1 when New is selected from the File menu brings up a help screen that explains what the New option is used for.

Another way to access the help system is to select the Help option at the right edge of the Editor bar menu. The Help menu is shown in Figure 7-14.

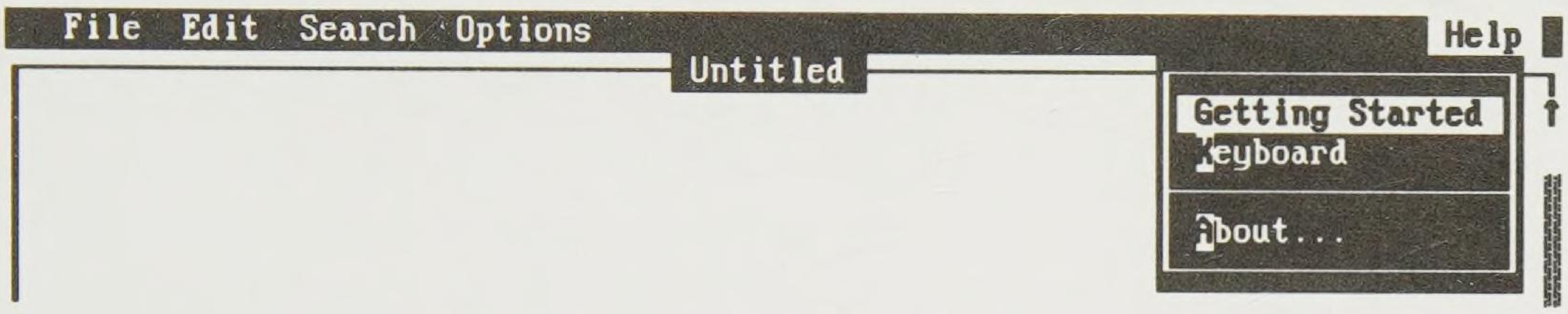


Figure 7-14 — Help Menu of MS-DOS Editor.

The Getting Started option brings up a dialog box that contains a general index of help topics. You may select any of these topics to see help on it. The Keyboard option brings up a dialog box that contains topics specifically related to the various keys used in the editor. The About option displays the version of the Editor that you are using and the copyright notice.

The third way to get online help is to select the Help button that appears on all dialog boxes. This produces a screen of help that explains the particular dialog box and what is expected of you when interacting with it.

One last point should be made about the MS-DOS Editor: it is essentially the same editor that Microsoft provides with its programming languages, including QBasic, which comes with DOS 5.0, and in fact the editor can only run when the file QBASIC.EXE is in the path or current directory, because the QBASIC program actually contains the editor within it. The EDIT command simply calls up a very small executable file named EDIT.COM, which in turns invokes the editor part of the QBASIC program. Consequently, you should not delete the QBASIC.EXE file from your disk, even if you do not expect to use the QBASIC language, unless you also do not want to use the editor.

Chapter 8

Batch Files

As you use your computer, you will find that you will execute certain sequences of commands over and over, particularly if you work frequently at the command prompt. For example, if your word processor document files are in a directory named C:\WP\DOCS, then to work on a document in your word processor you will need to enter a CD \WP\DOCS command to change to that directory, then the command that executes the word processor. When you are finished working in your word processor you may want to return to the root directory, which will require a CD \ command.

So there are two or three commands that you must remember and enter every time you edit a file in your word processor. Batch files are files that contain "batches" of DOS commands that can be used to automate these repetitive tasks. Batch files may contain any set of commands that you enter at the command prompt. These commands are then executed sequentially by simply entering the name of the batch file as though it were a command.

In addition to the usual DOS commands, there are some special commands for use only in batch files. These provide a limited form of programming power in batch files. This chapter will explore the creation and use of batch files. Familiarity with the DOS commands discussed in Chapters 3 and 5 is assumed.

A batch file is identified by the extension BAT. Batch files are executable files — DOS interprets their contents as instructions, and carries them out. When you type a command at the DOS prompt, DOS searches in the current directory and in any directories identified by the current path for a file that has the name you typed in, followed by an extension of COM, EXE or BAT. DOS will only attempt to execute files with these extensions, and it does so in a different way for each extension. You could not name a batch file with a COM extension and have it execute. Batch files must only have BAT extensions, and you should avoid naming batch files with the same name as programs that have COM or EXE extensions.

Most DOS commands may be included in batch files, as well as commands that run software programs. For example, if a word processor is started by typing WP at the DOS prompt, then the command WP may be included in a batch file. When DOS executes a batch file, it stores the entire file in memory until all of the commands have executed. This means you can include commands that will execute both before and after a program that is run from within a batch file.

Batch File Programs versus Shell Programs

With the ability to add menus to the Shell and write Shell programs, it may seem that batch programs are unnecessary. They have many of the same drawbacks as entering commands at the traditional DOS prompt — you must memorize their names, they have to be typed in from the keyboard, and when they require parameters there are no dialog boxes or help screens to assist you. For these reasons it may be desirable to use Shell programs rather than batch files whenever possible. However, batch files have some advantages too.

Shell programs are limited in the number of characters they may contain, and consequently in the number of commands that may be included in them. There is no such limit to the number of commands you may include in a batch file — you may make them as long as you like. You may also include remarks in batch files.

This makes the file much easier to read and understand, especially when you haven't looked at it for several months.

Shell programs are entered in one long line, and you can only look at a small part of the line at any time. This makes it difficult to follow a long sequence of commands. There are only very limited editing capabilities within the Shell for modifying program command lines, whereas a batch file can be edited with almost any word processor.

Finally, batch programs allow for certain very powerful programming statements that are unavailable to Shell programs. These are the IF command, which allows conditional processing; the GOTO command, which allows branching within a batch file; and the FOR command, which allows repetitive execution of one or more commands within a batch program.

Both batch files and Shell programs have their place. There are advantages to each of them. The best strategy is to use a Shell program where that is sufficient, and to use batch files where their additional power or size is required. When batch files are needed they can be called from within the Shell. This allows you to use the elegance of the Shell, with its menus and parameter dialog boxes, as a front end to the batch files, thereby eliminating the clumsiness of the traditional DOS interface.

Creating Simple Batch Files

The simplest batch files have only a few commands in them. A common application for a simple batch file is to change the current directory to one that contains a program you want to use, start the program, then when it is finished return to the root directory. For example, if the command that starts your word processor is WP, and you want to start it from the C:\WP\LETTERS directory, then the following batch file will take you to that directory, run the word processor, and return you to the root directory when you exit the word processor.

```
CD C:\WP\LETTERS

WP

CD \
```

Each command in a batch file appears on its own line. As each line executes it appears on the screen; this is called "echoing" — the commands are echoed on the screen as they are processed.

Using Echo and Pause Commands in Batch Files

The ECHO command is used in batch files to cause a message to display on the screen. The PAUSE command is used to suspend the batch file from executing until a key is pressed. ECHO has an additional use in batch files: it acts as a switch that either prevents or causes the commands within a batch file to appear on the screen as they execute. The following batch file demonstrates this:

```
ECHO OFF

CLS

ECHO Make sure the word processor disk is in drive A:

PAUSE

A:WP
```

If this batch file was named WP.BAT, entering the command WP would cause the screen to clear and the following two messages to appear:

```
Make sure the word processor disk is in drive A:

Press any key to continue . . .
```

After a key is pressed the WP program in drive A: will execute.

In this batch file, the ECHO OFF command turns off echoing. This means that as the subsequent batch file commands execute they are not displayed on the screen. However, any output they may produce does appear on the screen. The next command, CLS, causes the screen to clear. It is a good idea to start batch files with CLS in order to remove any distracting information that may still be there from the last command. This is not necessary for batch files that are called from a Shell program, since the Shell automatically clears the screen when it executes a program.

The next ECHO command is used to display a message on the screen — in this case, a prompt to put a disk in drive A. The PAUSE command causes DOS to stop executing the commands in the DOS file and display the message "Press any key to continue..." When you press a key, DOS resumes executing the batch file. The last command, (A:WP), executes a program named WP on the A: drive.

When a batch file finishes executing, DOS restores the ECHO state to whatever it was when the batch file was called. Thus, if ECHO is ON when a batch file is called, ECHO will be turned ON when the batch file finishes. If ECHO is OFF when the batch file is called it will be switched OFF when the batch file finishes.

Another use of the PAUSE command is to display more than one screen of information, but to permit the user to finish reading the first screen before the next one displays. For example, there may be two screens of help information that you want to make available to users. The information could be in two files, called HELP1.FIL and HELP2.FIL. The following batch file, named HELP.BAT, will display the help information one screen at a time, when the command HELP is entered:

```
CLS

TYPE HELP1.FIL | MORE

PAUSE

TYPE HELP2.FIL | MORE
```

The first TYPE command types the information in the file HELP1.FIL on the screen, and the MORE filter causes it to pause whenever the screen fills up. However, the last part of the HELP1.FIL file could be scrolled off the screen without a pause by the start of the second TYPE command. So the PAUSE command is used to halt the execution of the batch file after the first file is displayed, and wait for you to press a key. DOS then goes on to display the second screen of information.

Parameters and Environment Variables in Batch Files

The term "parameters" refers to variables in your batch files — items that are referenced within the file, but whose specific value can change each time the file is run. A batch file that clears the screen and then lists the WP directory on drive C could contain the following two commands:

```
CLS
DIR C:\WP /P
```

If this file was named D.BAT, you could simply type the letter D to list the files in the C:\WP directory. However, if you wanted to list the files in different directories, you would need a separate batch file for each directory, and you would have to remember the name of each file. It would probably be easier to just type the DIR command each time, specifying the directory you want to see.

A better solution is to use a batch file that accepts a parameter to tell it which directory to list. The following batch file demonstrates this:

```
CLS
DIR %1 /p
```

Batch file parameters are represented within the batch file by the % sign followed by a number from 1 to 9. When the file is executed, each parameter is replaced by the corresponding item from the DOS command that called the file. For example, if the above two commands were in a file named D.BAT, then entering the command D C:\WP would cause the %1 to be replaced by the letters C:\WP. This results in the command DIR C:\WP /P being executed. The next time the file is run, it can be entered as D A:, which displays the directory of drive A. More than nine parameters may be referenced in a batch file, with some limitations, using the SHIFT command. SHIFT is discussed in Chapter 10.

An example of a batch file using two parameters follows. It creates a new directory, then copies files from an existing directory to the new one.

```
MD %1
COPY %2 %1
```

If this batch file is named MC.BAT, the following command will create a new directory named NEWDIR and copy all of the batch files from the C:\ directory into the NEWDIR directory:

```
MC NEWDIR C:\*.BAT
```

Environment variables may be referenced from batch files, by enclosing them in percent signs (%). In Chapter 5, the following batch file was presented as a means of appending a new directory onto an existing path:

```
SET PATH1=%PATH%
PATH=%PATH%;%1
```

This batch file refers to the PATH environment variable, by enclosing it with % percent signs. The first line sets a new environment variable, named PATH1, to contain the existing path, and the second line appends the first parameter passed to the batch file onto the end of the existing path.

Creating Batch Files that Call Themselves

The names by which batch files refer to parameters start with %1 and end with %9. However, you may also use %0 in your batch files. %0 is a special parameter symbol that refers to the name of the batch file in which it occurs. This allows you to call a batch file from within itself. It might not seem useful at first to do this, but there are circumstances in which it can be a very efficient way to accomplish a task which must be repeated many times. The following batch file demonstrates this:

```
ECHO OFF
ECHO Place source disk in drive A and target disk in drive B
PAUSE
COPY A: B:
%0
```

If this file is called CCOPY.BAT, for Continuous COPY (note that you should not give batch files the same name as DOS commands, in this case COPY), then entering the command CCOPY will result in the message, "Place source disk in drive A and target disk in drive B" being displayed on the screen. Below that will be the message, "Press any key to continue . . ." After you have placed the disks in drives A and B and pressed a key, all of the files will be copied from drive A to drive B.

The %0 command will cause this sequence to repeat and the message "Place source disk in drive A and target disk in drive B" will again be shown, followed by the "Press any key to continue . . ." message. The batch file will continue executing like this until you press Ctrl-break. This will always cause a batch file to stop execution. Pressing Ctrl-break will cause the following prompt to be displayed:

```
"Terminate batch job (Y/N)?"
```

Pressing Y will return you to the DOS prompt and the screen should look similar to that shown in Figure 8-1.

```
Place source disk in drive A and target disk in drive B
Strike a key when ready . . .
A:JOHN.LET
A:MARY.LET
A:BILL.LET
A:GEORGE.LET
   5 File(s) copied
Place source disk in drive A and target disk in drive B
Strike a key when ready . . .
A:ACCNTS1.RBF
A:ACCNTS2.RBF
A:ACCNTS3.RBF
   3 File(s) copied
Place source disk in drive A and target disk in drive B
Strike a key when ready . . .
A:JULY.WK2
A:AUGUST.WK2
A:SEPT.WK2
2 File(s) copied
Place source disk in drive A and target disk in drive B
Strike a key when ready . . . ^C

Terminate batch job (Y/N)? y

C:\>_
```

Figure 8-1 —Result of Running CCopy Batch File.

Halting the Execution of a Batch File

You may encounter situations in which you want to stop a batch file before it is finished executing. This could be because you realize after starting it that it is going to take too long to complete, or that it is going to do something you do not want it to do. You may even have created a batch file that never terminates on its own, such as the previous example for copying large numbers of disks from drive A to drive B. Batch file execution may always be stopped by holding down Ctrl and pressing break. This is known as a "control-break," a special signal that DOS recognizes as a command to stop whatever it is doing and return to the DOS prompt.

Use of the IF Command in Batch Files

One of the powerful features of DOS batch files is the ability to execute a command only if some condition is true, or alternately if some condition is not true. Conditions that can be tested are:

1. The existence or non-existence of a file that is named in the batch file.
2. The existence or non-existence of a file passed as a parameter.
3. Whether a parameter matches a value that is specified in the batch file.
4. Whether or not a previous command terminated normally.

1. This first condition can be demonstrated with a batch file that displays the directory information for a file named TEST.FIL only if the file exists in the current directory. This file contains the following commands:

```
ECHO OFF
CLS IF EXIST TEST.FIL DIR TEST.FIL
IF NOT EXIST TEST.FIL ECHO Test.fil Does Not Exist
```

Note that echo is turned off with the ECHO OFF command, and the screen is cleared with the CLS command, at the start of this batch file. It can be desirable to have commands display on the screen when a batch file executes, but in general it is good style to not display them. They tend to be confusing, especially since they often flash by too quickly to read. Clearing the screen is usually a good idea because it removes clutter that makes it difficult to locate important messages or other information.

This file demonstrates the use of the NOT operator. You may place NOT in front of any of the IF condition tests in batch files. The effect is to test for the failure of the condition, rather than its success. When this file is executed, if TEST.FIL exists in the current directory the DIR listing for it will be displayed. If it does not exist, the message, "`Test.fil Does Not Exist`" will be shown instead.

2. Checking for the existence of a file name entered as a parameter can be demonstrated with a batch file that starts a word processor named WP with the specified file loaded, if the file exists. If the file does not exist the batch file displays an error message. The following batch file demonstrates:

```
ECHO OFF
CLS
CD C:\WP\LETTERS
IF EXIST %1 WP %1 IF NOT EXIST %1 ECHO %1 Does Not Exist - Check The Name
CD C:\
```

This batch file changes directories to the word processing directory C:\WP\LETTERS before checking for the existence of the file that is passed as a parameter. It is important to include a CD (Change Directory) command in batch files that serve the type of function demonstrated here, since the batch files are usually in the root directory and the program and data files are generally in subdirectories. It is also a good idea to return to the root directory at the end of the batch file, as is done here with the CD C:\ command, since that is where other batch files you might want to execute will most likely be found.

If this batch file is named WP.BAT, and a file named MYFILE.LET exists in the C:\WP\LETTERS directory, then the following command will result in the word processor WP starting up with the file MYFILE.LET loaded:

```
WP MYFILE.LET
```

If MYFILE.LET does not exist in the C:\WP\LETTERS directory, the message "MYFILE.LET Does Not Exist - Check The Name" will be displayed.

3. Testing whether a parameter matches a value that is included in the batch file can be demonstrated with a batch file that carries out one command if the name Mary is entered as a parameter, and another command if the name Bill is entered. The commands for this batch file are:

```
IF %1 == Mary WP
IF %1 == Bill DB
ECHO A valid name was not entered. Try again.
```

If this batch file is named NAME.BAT, then the following command will cause a program named WP to be run:

```
NAME MARY
```

If the name entered is neither Mary nor Bill, the ECHO command will execute, displaying the message "A valid name was not entered. Try again." Notice that there are two equal signs in these condition tests. If you use only one equal sign, DOS will terminate execution of the batch file and display the message "Syntax error."

The WP and DB programs could be batch files, or they could be EXE or COM files, but, they could not be DOSKEY macros. Macros cannot be run from within a batch file, though batch programs can be run from within macros. If WP and DB were COM or EXE files, then the ECHO command would always execute. This is because when the WP or DB programs ended, control would return to the NAME.BAT file, and the ECHO command would execute before the batch file terminated. However, when one batch files executes a second batch file in the manner demonstrated here, control will not return to the first batch file. If you want control to return to a batch file after it runs another batch file you must use the CALL command, discussed later in this chapter.

The names Mary and Bill must be spelled exactly the same on the parameter line and in the batch file. This includes case sensitivity. If you entered the command NAME MARY or NAME mary, the IF condition would fail and the ECHO command would execute.

4. Testing for normal termination of the previous command only works with certain DOS commands and certain programs. When these commands terminate, they leave a code known as the EXIT CODE in the DOS environment

ERRORLEVEL variable. Batch files may access this code to determine the status of the last command's termination — whether it terminated normally or abnormally.

Usually an exit code of 0 means the command terminated normally — it carried out its intended purpose with no interruption or error. A non-zero exit code will indicate an abnormal termination. This can be caused by the user interrupting the operation or by an error occurring during the operation. The exit code will indicate what caused the interruption. Generally, the higher the exit code the more serious the problem. DOS exit codes are shown in the Chapter 10, for those DOS commands that generate exit codes.

One of the DOS commands that sets an EXIT code is the BACKUP command (see Chapter 6). This command is used to back up a disk, usually a hard disk, onto one or more floppy disks. The following batch file tests for an exit code from the BACKUP command and takes an action based on that code.

```
BACKUP %1
IF ERRORLEVEL 0 ECHO Backup finished normally.
IF ERRORLEVEL 1 ECHO No files were found for backing up.
IF ERRORLEVEL 1 ECHO Check the file specification and try again.
IF ERRORLEVEL 2 TYPE BUERR.2
IF ERRORLEVEL 3 ECHO You have terminated the backup.
IF ERRORLEVEL 4 TYPE BUERR.4
```

This file relies on two text files, BUERR.2 and BUERR.4, that could contain lengthy error messages and instructions if the BACKUP command returns an exit code of either 2 or 4.

Placing Comments in Batch Files

As batch files become larger and more complex, they also become harder to understand. The REM command helps make batch files easier to understand. REM stands for "remark." When a batch file line begins with the letters REM, everything on that line is ignored when the batch file executes. The line is used for a comment that explains something about the batch file. REM lines can also be used to separate commands from one another, making the batch file easier to read.

Remarks will be echoed to the screen if ECHO is set on, so it is a good idea to always start your batch files with the ECHO OFF command. If there are particular lines in your file you want echoed, you can turn ECHO back on for those lines. Many of the examples in the rest of this chapter will make use of REM lines to help explain how the files work.

Developing Complex Batch Files

To demonstrate more sophisticated batch file techniques, an online help system for three programs, a word processor, a spreadsheet, and a database program, will be developed entirely with batch files. This system will use three files to contain help information for the respective programs. The file that contains the help for word processing should be named WP1.FIL; the file that contains the help for the database system should be called DB1.FIL; and the file that contains the help for the spreadsheet system should be called SS1.FIL. Each of these files should contain less than one screen of text. If you want to provide more than one screen of information for a subject, enter the second screen into an xx2.FIL file — a second screen of information for the spreadsheet goes in the SS2.FIL. A third screen would go in the SS3.FIL, and so on.

The name of the online help batch file is HELP.BAT. Figure 8-2 contains the contents of HELP.BAT.

```
@ECHO OFF
REM filename: HELP.BAT
REM This file is used to display help screens that are stored
REM in files named XX1.FIL, XX2.FIL, and so on.  The XX's stand
REM for the application for which the help is provided. For
REM example, the word processor help screens are stored in files
REM named WP1.FIL, and WP2.FIL.  This file is executed by entering
REM the command HELP XX - e.g. HELP WP
CLS
IF NOT EXIST %11.FIL ECHO No help files for that command.
IF EXIST %11.FIL TYPE %11.FIL
IF EXIST %12.FIL PAUSE
IF EXIST %12.FIL TYPE %12.FIL
IF EXIST %13.FIL PAUSE
IF EXIST %13.FIL TYPE %13.FIL
```

Figure 8-2 — Batch File for an Online Help System.

To use this help system for assistance with starting the word processing system, a user would enter the command HELP WP. The HELP.BAT file then checks for the existence of a file named WP1.FIL. If the file does not exist, it will display the message "No help files for that command." If the file does exist, HELP.BAT will display the contents of the file on the screen, then check for the existence of a file named WP2.FIL. If WP2.FIL exists, the PAUSE command will execute, displaying the message "Press any key to continue . . ." on the screen. When a key is pressed the contents of WP2.FIL will be displayed. The process will then repeat for the file named WP3.FIL.

This system may be expanded to provide as many HELP screens as you like, for as many applications as you like. You must make sure that you allow for the maximum number of help files you have written for any one of the commands. If you have five help files for the database system (DB1.FIL, DB2.FIL, DB3.FIL, DB4.FIL and DB5.FIL), and three for each of the others, then you must have IF EXIST commands for five files.

If you have been typing in and executing the examples up to now, you may have noticed that whenever there is an ECHO OFF command, it echoes on the screen. This makes sense, since ECHO is ON when the command is executed. But it can also be annoying because one of the goals is to make the batch file commands transparent to the end user. You can't do that if they echo on the screen. The ECHO OFF command in the example for the online help system did not appear on the screen. The reason is that it begins with an @ sign. The @ sign tells DOS not to echo the command on the screen, regardless of the ECHO status. You can use this in front of any batch command to selectively suppress it from echoing.

This Help system demonstrates the ability to insert a parameter into a text string. When DOS executes a batch file that contains parameters it literally inserts the parameter values directly into the command.

Branching of Program Control with the GOTO Command

The GOTO command can be used in conjunction with the IF command to add great flexibility and power to batch files. GOTO causes control to jump from one line in the file to another. It is used in conjunction with a label that indicates the point in the file to jump to. For example, the batch file in Figure 8-3 uses a GOTO to display one message if the file passed as a parameter exists, and another message if the file does not exist.

```
1   @ECHO OFF
2   REM This file is named JUMP.BAT.  It tells the user whether
3   REM or not the file passed as a parameter exists.
4   REM
5   IF NOT EXIST %1 GOTO NOFILE
6   ECHO The file %1 exists.
7   GOTO :END
8   :NOFILE
9   ECHO The file %1 does not exist.
10  :END

(Line numbers are for reference only; they should not
be included in the batch file.)
```

Figure 8-3 — Batch File with GOTO Command.

This batch file first checks to see if the file passed as a parameter exists. If it does, he GOTO on line 5 does not execute and control continues with line 6. This displays the message, "The file (filename) exists." Then the GOTO on line 7 executes, causing control to jump past lines 8 and 9 to line 10, which is the end of the file.

On the other hand, if the file that is passed as a parameter does not exist, then the GOTO on line 5 executes, causing control to jump past lines 6 and 7 and continue on line 9 (line 8 is a label line, which does not execute — it simply marks the point to jump to). This displays the error message, "The file (filename) does not exist."

The line on which the label appears must begin with a colon (:), and the label must be the only thing that appears on the line after the colon. Lines containing a label in a batch file are not echoed on the screen, even if ECHO is set to on. This provides an alternative to the REM command for including remarks in a batch file: start the line with a colon, which identifies it as a label to DOS, then place your remark after the colon. The line will not be echoed on the screen, as REM lines are if ECHO is on. The only thing you must guard against if you use colons instead of REM is having a comment that matches a label. This may cause control to jump to the comment, rather than the true label.

The batch file in Figure 8-4 sets up the same online help system described in Figure 8-2, only Figure 8-4 uses the GOTO command to reduce the number of IF commands.

```
@ECHO OFF
REM filename: HELP.BAT
REM This file is used to display help screens that are stored
REM in files named XX1.FIL, XX2.FIL, and so on.  The XX's stand
REM for the application for which the help is provided.  For
REM example, the word processor help screens are stored in files
REM named WP1.FIL and WP2.FIL.  This file is executed by entering
REM the command HELP XX - e.g. HELP WP
CLS
IF NOT EXIST %11.FIL GOTO NOFILE
TYPE %11.FIL
IF NOT EXIST %12.FIL GOTO END
PAUSE
TYPE %12.FIL
IF NOT EXIST %13.FIL GOTO END
PAUSE
TYPE %13.FIL
:NOFILE
ECHO No help files for that command.
:END
```

Figure 8-4 — Use Of GOTO to Improve the Online Help System.

Using FOR to Repeat Commands

The FOR command is used in batch files to cause a command to repeat a specified number of times. It does this by specifying a variable that takes on each of the values in a set that is specified in the command. This is similar to the use of the FOR command discussed in Chapter 3. The variable is named with two percent signs followed by any one character, for example %%A. The set of values which the variable will take on is specified by placing them within parentheses in the FOR command. Figure 8-5 contains a simple batch file, named SHOW.BAT, that uses the FOR command.

```
@ECHO OFF
REM This file is named SHOW.BAT.  It is used to display the
REM directory listing of the three files entered in the
REM parentheses.
REM
FOR %%F IN (FILE1 FILE2 FILE3) DO DIR %%F
```

Figure 8-5 — Use of the FOR Command.

When the SHOW.BAT file is called, the FOR command gets translated by DOS into three separate commands that look like this:

```
DIR FILE1

DIR FILE2

DIR FILE3
```

The variable %%F takes on each value in the set (FILE1 FILE2 FILE3), and is passed to the DIR command for each of these values. This may not seem useful at first, but the values in parentheses (called the SET of the FOR command) can be replaced by parameters passed to the batch file on the command line. Figure 8-5 could be rewritten as in Figure 8-6.

```
@ECHO OFF
REM This file is named SHOW.BAT.  It is used to display the
REM directory listing of the three files entered in the
REM parentheses.
REM
FOR %%F IN (%1 %2 %3) DO DIR %%F
```

Figure 8-6 — Use of Parameters with the FOR Command.

With the SHOW.BAT file shown in Figure 8-6, you could enter the command SHOW FILE1 FILE2 FILE3, and DOS would display the directory listing for each of the three files specified on the command line. You could take this a step further and, as shown in Figure 8-7, and display messages about files that do not exist, and display the directory listing for files that do exist, all with only two commands.

```
@ECHO OFF
REM This file is named CHECK.BAT.  It is used to display the
REM directory listing for files that are entered on the command
REM line.  If a file does not exist, a message is displayed that
REM lets the user know a non-existent file name was entered.
REM
FOR %%F IN (%1 %2 %3 %4) DO IF EXIST %%F DIR %%F
FOR %%F IN (%1 %2 %3 %4) DO IF NOT EXIST %%F ECHO %%F does not exist
```

Figure 8-7 — Combining FOR and IF Commands.

If the CHECK.BAT file shown in Figure 8-7 were executed with the command CHECK FILE1 FILE2 FILE3, and only FILE2 existed, the result would appear similar to the following:

```
C:\DOS>TEST FILE1 FILE2 FILE3
   Volume in drive C is DOS 500
   Volume Serial Number is 1A5A-20AE
Directory of C:\DOS
FILE2         7 06-01-91  5:23p
        1 File(s)   9166848 bytes free
FILE1 does not exist
FILE3 does not exist
C:\DOS
```

Notice that only three file names were entered, even though the batch file allows for four parameters. The fourth parameter is ignored if nothing is entered for it, so you should make sure you allow for the maximum number of parameters in the FOR command set.

The online help system shown in Figures 8-2 and 8-4 can be greatly simplified using a single FOR command and two batch files. The batch files are shown in Figures 8-8 and 8-9.

```
@ECHO OFF
REM This file is named HELP.BAT.  It is used to display help
REM files that have been prepared for one of the applications
REM available to users of this computer.  There can be up to
REM four help files per application.  This number can be
REM increased simply by adding additional parameters to the
REM FOR command set.
REM
CLS
FOR %%F IN (%11.FIL %12.FIL %13.FIL %14.FIL) DO CALL SHOW %%F
```

Figure 8-8 — Use of FOR Command to Improve Online Help.

```
REM This file is named SHOW.BAT.  It is called from the file
REM HELP.BAT.  It displays the help screen file that is passed
REM to it as a parameter, then pauses so the user will have time
REM to read the screen.  It checks for the existence of the file
REM because HELP.BAT is going to pass four file names, even if
REM they don't all exist.  Checking for their existence prevents
REM the appearance of extraneous messages appearing on the
REM screen.
REM
IF EXIST %1 TYPE %1
IF EXIST %1 PAUSE
```

Figure 8-9 — SHOW.BAT File Called from HELP.BAT.

After you take out the remarks, there are only five commands in these two files taken together, as opposed to eight and 12 commands in Figures 8-2 and 8-4, respectively. The CALL command in HELP.BAT is discussed in the next section.

The most important improvement in the system shown in Figures 8-8 and 8-9 is that it is much easier to understand and modify. If you want to add additional help files you simply increase the number of parameters in the set of the FOR command.

As you can see, the FOR command can be very useful for concisely repeating the processing of a DOS command or program file. You can even use it to cause another batch file to execute a specified number of times, without passing any additional parameters to the batch file. A batch file that uses the FOR command in this way is shown in Figure 8-10.

```
@ECHO OFF
REM This file is named REPEAT.BAT.  It calls the batch file
REM NEXT.BAT 10 times.  This number may be adjusted simply by
REM changing the numbers in the set of the FOR command.
REM
FOR %%N IN (1 2 3 4 5 6 7 8 9 10) DO CALL NEXT
```

Figure 8-10 — Use of the FOR Command to Repetitively Execute a DOS Command.

This file will call a batch file named NEXT.BAT 10 times.

Use of the FOR command at the DOS prompt is discussed in Chapter 3.

Batch Files that Call Other Batch Files

As seen in some of the examples above, a batch file can cause another batch file to execute simply by having the name of the second batch occur as a command in the first. When this happens, the first batch file gives up control to the second one. Any commands that appear in the first batch file after the command that calls the second batch file will not execute, unless there is a GOTO that can cause control to branch around the reference to the second batch file. The batch files in Figures 8-11 and 8-12 demonstrate this.

```
@ECHO OFF
REM This file is named GREETING.BAT
REM
ECHO Hello.
FILE2
ECHO Goodbye.
```

Figure 8-11 — GREETING.BAT File with Nonreturnable Call to RESPONSE.BAT.

```
REM This file is named RESPONSE.BAT
REM
ECHO You'll never say goodbye.
```

Figure 8-12 — RESPONSE.BAT File.

When the command GREETING is entered, the following two messages will appear on the screen:

```
Hello.
You'll never say goodbye.
```

This happens because when GREETING passes control to RESPONSE, it gives up control altogether. There is no way for the `ECHO Goodbye` command to execute. If you want control to return to the calling file you must execute the second batch file with a CALL command. When the second batch file finishes executing, control then returns to the first batch file, at the line immediately following the CALL command. The batch file in Figure 8-13 illustrates this.

```
@ECHO OFF
REM This file is named GREETING.BAT
REM
ECHO Hello.
CALL RESPONSE
ECHO Want to bet?
ECHO Goodbye!
```

Figure 8-13 — GREETING.BAT File with Returnable Call to RESPONSE.BAT.

Now when the command GREETING is entered, the following messages appear on the screen:

```
Hello.
You'll never say goodbye.
Want to bet?
Goodbye!
```

As you can see, the CALL command causes control to return to the calling file. You may also have noticed that there is no ECHO OFF command at the beginning of RESPONSE.BAT. The ECHO state is passed from one batch file to another, and restored when control returns. This means that when GREETING.BAT sets ECHO OFF, it remains off in RESPONSE.BAT unless an ECHO ON command is encountered. If RESPONSE.BAT sets ECHO ON, it will be restored to its off status when control returns to GREETING.BAT.

You may pass parameters from one batch file to another. For example, the batch files in Figures 8-14 and 8-15 both accept parameters. The file in Figure 8-14 calls the file in Figure 8-15, passing its second parameter to the called batch file's parameter number one:

```
@ECHO OFF
REM This file is named BATCH1.BAT
COPY %1 %2
BATCH2 %2
```

Figure 8-14 — Passing Parameters from One Batch File to Another.

```
REM This file is named BATCH2.BAT
DIR %1
```

Figure 8-15 — Batch File that Receives Parameters from Another Batch File.

The first file, BATCH1, can be called with the following command:

```
BATCH1 TEST.FIL C:\DOS
```

This will copy the file named TEST.FIL to the DOS directory. TEST.FIL will become parameter %1 in this file and C:\DOS will become parameter %2. The second batch file will then be called with the command BATCH2 C:\DOS, which will result in the C:\DOS directory being displayed.

Batch files that call each other may be nested as deep as you like — there is no limit to how many you may have. You may even have a batch file that calls itself, as demonstrated earlier, or two batch files that call each other. However, there are a few important considerations to keep in mind.

Each time a batch file calls another batch file, the first file must be stored in memory until control returns to it, so that DOS will know what command to execute at that time. If the second batch file calls a third batch file, the second batch file as well must be stored in memory. Eventually all of the available memory could be used up and the computer will lock up and stop responding to the keyboard. You will have to re-boot in order to get started again. Consequently, if you are going to use the CALL command you must think through its logic very carefully, to make sure that control will eventually be passed back to the first execution of the first file in the calling sequence, so the whole process can terminate.

If you suspect that a batch file sequence has gone into an endless loop, or is in danger of running out of memory, you may always stop it with Ctrl-break.

There is an important difference between using CALL to call another batch file, and using %0 in a batch file to recursively reference itself. Since no CALL command is involved with %0, DOS does not save the calling file in memory in order to return control to it. The calling file is discarded when the %0 command is encountered, so there is no danger of running out of memory.

Chapter 9

Configuring and Customizing DOS

DOS commands can be thought of as falling into two broad categories: commands that do some kind of work for you, and configuration commands. Most of the commands discussed up to this point fall into the former category. Some were configuration commands, but they related to very specific aspects of DOS, which warranted their own chapter (such as memory configuration commands). The rest of the configuration commands do not fit neatly into one subject area, but instead comprise a sort of grab bag of functions that influence various aspects of DOS. These commands are the subject of this chapter.

The CONFIG.SYS and AUTOEXEC.BAT Files

The first thing DOS does when you start your computer is read the contents of the CONFIG.SYS file and execute the commands it finds there. The second thing it does is read the AUTOEXEC.BAT file and execute the commands it finds there. The CONFIG.SYS file contains commands that can generally be executed only at boot time, while the commands in AUTOEXEC.BAT are standard DOS commands that could be entered at the command prompt. Configuring DOS consists primarily of altering the contents of these two files, though some commands that can be thought of as configuration commands, like EMM386 and DOSKEY, can be entered several times during a session.

The DOS Environment

When the computer is booted, DOS sets up an area of memory known as the "environment." This is an area that contains common values that DOS and some software programs will need to find. The format of the DOS prompt is stored in the environment, as is the current path. In addition, the environment contains variables, called "environment variables." They are called environment variables because they are stored in the environment space, and they can be created, modified, and dropped at will. An environment variable that is used with DOS 5.0 is DIRCMD, which is used to define default settings for the DIR command. DIRCMD is discussed in more detail in Chapter 5.

A common use of an environment variable by a program is to store the name of a directory that a program needs to reference. Rather than force you to use a particular name for a directory, the program will simply require that you put the name of whatever directory you want to use in an environment variable. The program then refers to that environment variable when it needs to find the directory. For example, the Microsoft C language compiler needs to know where a set of files known as "library files" are stored. The instructions for the compiler tell you that you may use whatever directory you want, but you should enter the following command before starting the compiler:

```
SET LIB=dirname
```

Where **dirname** is the complete path specification for the directory in which the library files will be stored. If you wanted to store them in the C:\C600\LIB directory, then you would enter this command:

```
SET LIB=C:\C600\LIB
```

This allows you to name your directories whatever you want, and still let the compiler know where the library files are stored.

Environment variables may be referenced within batch files, by surrounding them with percent signs. An example of this can be seen in Chapter 8.

Creating Environment Variables with the SET Command

As seen above, the SET command is used to define environment variables, and to change or remove them. When SET is entered by itself, it displays the current environment. A typical environment displayed by the SET command is shown in Figure 9-1.

```
COMSPEC=C:\DOS\COMMAND.COM
PROMPT=$P$G
PATH=C:\DOS;C:\RBFILES;C:\WP;C:\GV;
DIRCMD=/P
```

Figure 9-1 — Typical DOS Environment as Displayed by SET Command.

This environment has an environment variable named COMSPEC, which contains the path to the file COMMAND.COM. Under some circumstances, it is necessary for DOS to reload part of the COMMAND.COM file because it was overwritten in memory. The COMSPEC environment variable tells DOS where to find the COMMAND.COM file. The next line shows the current definition of the prompt, followed by the current path. The last line contains the definition of the DIRCMD environment variable. The path and prompt appear to be the same as environment variables, and indeed they are — when DOS needs to know how one of these is defined, it looks up the appropriate environment variable.

The SET command creates new environment variables by following the word SET with an equal sign, then a text string that should be contained in the variable. There can be no spaces on either side of the equal sign.

To remove an environment variable, enter the SET command with nothing following the equal sign. For example, to remove the DIRCMD environment variable, enter the following command:

```
SET DIRCMD=
```

Environment variables remain in the environment until they are either removed in this way, or the computer is turned off. These variables take up memory, and there is only a specific amount of memory available for the environment. If you attempt to create an environment variable, or do something else that requires environment space, like defining a larger path or prompt, and receive the message "Out of environment space," it means that all of the memory reserved for the environment is being used. Removing some environment variables may fix the problem. If you need more environment space than you have, you may specify more with the SHELL command.

Changing the Environment Size with SHELL

The SHELL command defines the command interpreter that DOS uses to interpret the commands entered on the command line. The parameters and switches used with SHELL follow the name of the command interpreter, and are actually parameters and switches that the command interpreter program accepts, rather than SHELL. The standard DOS command interpreter is COMMAND.COM, and if no SHELL command is found in the CONFIG.SYS file, DOS automatically looks for COMMAND.COM in the root directory of the boot disk.

SHELL can also be used to change the amount of memory available for the DOS environment. Note that SHELL is a CONFIG.SYS command, and only executes at boot time. Its effects cannot be changed or undone except by modifying the CONFIG.SYS file and rebooting the computer. The DOS environment is normally set to 256 bytes. To change the size of the environment, use a SHELL command similar to the following:

```
SHELL=C:\DOS\COMMAND.COM /E:xxxx /P
```

Whenever DOS sees a SHELL command in the CONFIG.SYS file, it expects to be told what command interpreter to use. The command interpreter must, therefore, be fully defined (path and file name). In this case, the command interpreter is COMMAND.COM, located on the C drive in the DOS directory. The /E switch specifies the environment size, where **xxxx** specifies the size in bytes. The range for this value is between 160 and 32768. If the number entered is not a multiple of 16, DOS rounds it up to the next higher multiple of 16. Therefore, it is a good idea to always enter a number that is a multiple of 16, so that you know exactly how much memory is allocated to the environment. The size of the environment should be large enough to store all of the environment variables that you need, but no more. Memory allocated to the environment is taken away from the memory available for running programs.

Because the SHELL command is included in the CONFIG.SYS file, the /P switch should always be used. /P causes the copy of COMMAND.COM to remain in memory, and causes the AUTOEXEC.BAT file to execute.

SHELL can be used to define a different command interpreter, if you do not want to use COMMAND.COM. This is a rare occurrence, but if you want to do it, all you have to do is include the following line in your CONFIG.SYS file:

```
SHELL=path\filename
```

where **path\filename** represents the complete path and file name of the command interpreter you want to use.

Device Drivers

Any piece of hardware that is attached to the computer, and provides some form of input or receives some form of output, is referred to as a "device." Each device is unique, and requires its own software, called a "device driver" to manage its communications with the computer. Some device drivers are built-in, and others can be installed by including them in the CONFIG.SYS file, using the DEVICE= command. The DEVICE= command was explained in Chapter 5, where it was used to install the HIMEM.SYS and EMM386.EXE device drivers, and in Chapter 6, where it was used to install the RAMDRIVE.SYS and SMARDRV.SYS device drivers. There are several other standard device drivers provided with DOS 5.0: ANSI.SYS, DISPLAY.SYS, DRIVER.SYS, EGA.SYS, and PRINTER.SYS.

STOP

There are two files provided with DOS 5.0 that would appear from their names to be device drivers: COUNTRY.SYS and KEYBOARD.SYS. These are not device drivers, and attempting to load them with the DEVICE= command will lock up your computer. Since the command is in the CONFIG.SYS file, you will have to boot the computer with a different disk and edit CONFIG.SYS before that disk can be used again to boot the computer. The COUNTRY= and KEYBOARD= commands will load these SYS files as they need them.

Using ANSI.SYS to Customize the Screen and Keyboard

One of the standard device drivers provided with DOS 5.0 is the ANSI.SYS driver. This driver allows you to modify the way the CPU responds to the keyboard and the way it displays characters on the screen. One of the ways of sending commands to this driver is through the PROMPT command, using the $E parameter. $E inserts into the command line a character known as the Escape character. The Escape character tells DOS that what follows is one or more commands that should be sent to the ANSI.SYS device driver. The combination of the Escape character and ANSI.SYS commands is known as an “Escape Sequence.”

It is important to keep in mind how these escape sequences work. They are not really PROMPT commands, rather they use the PROMPT command to send a command sequence to the ANSI.SYS driver. For this reason, entering another PROMPT command does not automatically cancel them, the way it would cancel previous PROMPT commands. The only time a PROMPT command communicates with the ANSI.SYS device driver is when an escape sequence is included in the command, and the only time a previous command to ANSI.SYS is canceled is when a specific cancellation command is sent to ANSI.SYS.

The ANSI.SYS device driver can also receive commands from programs written in programming languages like BASIC, PASCAL, and C, but a discussion of these uses is beyond the scope of this book.

PROMPT Parameters Used for Positioning the Cursor

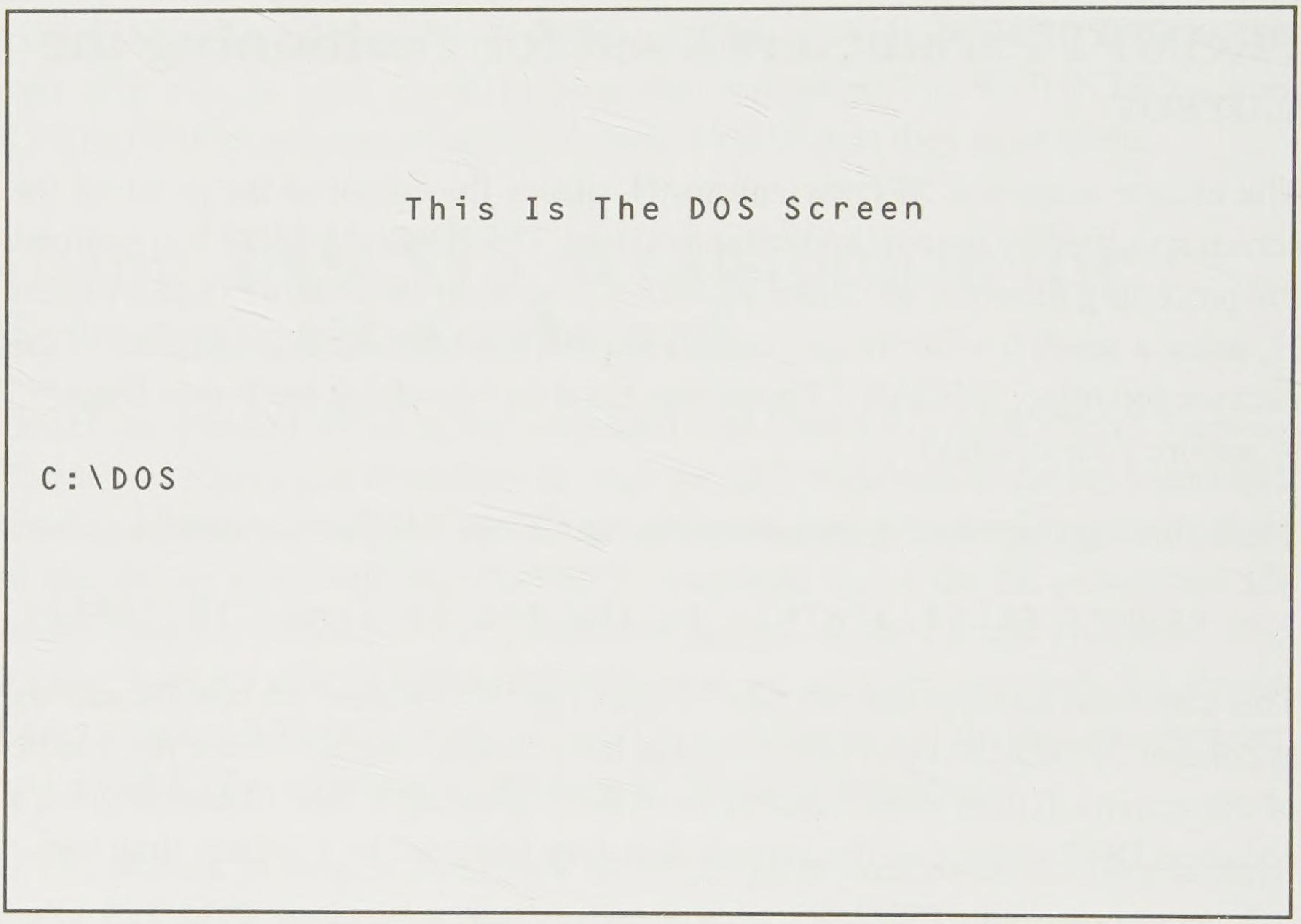
The escape sequence **$E[row;columnH** places the cursor at the point on the screen specified by the row and column values. The **H** tells ANSI.SYS to interpret the preceding numbers as cursor positions. You must be certain to use a capital H, using a small h will change your screen display. (If this happens, refer to the next section titled, "PROMPT Parameters Used for Modifying the Screen Display" to restore your display).

The following command demonstrates the use of PROMPT to position the cursor:

```
PROMPT $E[10;31HThis Is The DOS Screen$E[12;1H$P$G
```

This command displays the message "This Is The DOS Screen" on row 10, starting at column 31 (which centers the message horizontally, slightly above the middle of the screen). It then moves the cursor to the beginning of line 12 and displays a common DOS prompt — the current directory followed by a greater than sign.

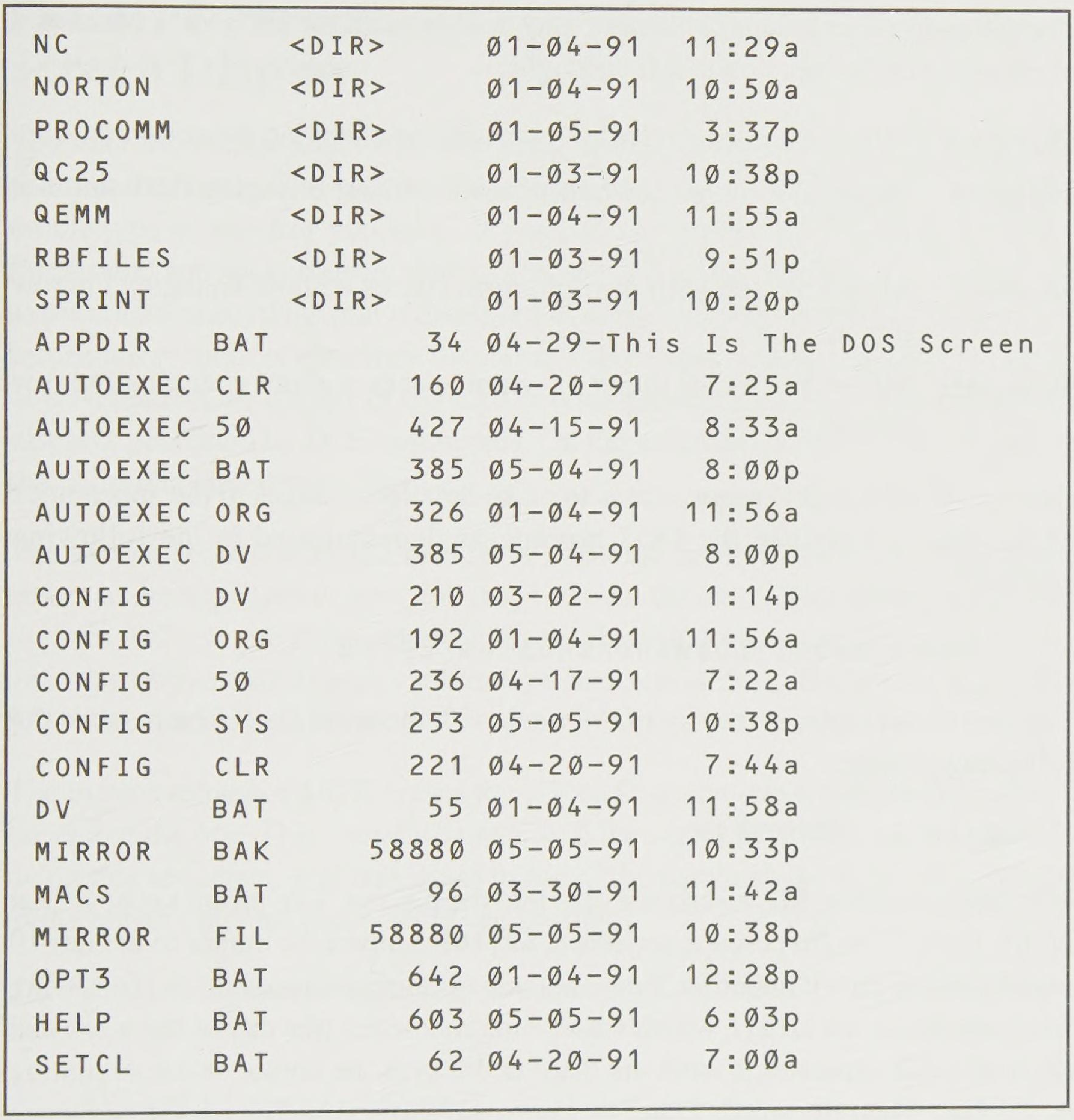
This prompt appears at the specified location, regardless of whatever else is on the screen. It does not clear the screen first, so the prompt may appear in the middle of other information already being displayed. Whether this is a problem depends on your anticipated use of the DOS command line. You may clear the screen before the message is displayed, or only clear the lines on which the prompt will appear, using the escape sequences discussed in the next section. If you enter the CLS command, which clears the screen, after entering this PROMPT command, the screen will appear as in Figure 9-2.



```
                This Is The DOS Screen




C:\DOS
```

Figure 9-2 — DOS Prompt Using ANSI.SYS for Cursor Positioning.

If the screen is not cleared first, the message could appear in the middle of information displayed by the last command, as shown in Figure 9-3.



```
NC           <DIR>     01-04-91  11:29a
NORTON       <DIR>     01-04-91  10:50a
PROCOMM      <DIR>     01-05-91   3:37p
QC25         <DIR>     01-03-91  10:38p
QEMM         <DIR>     01-04-91  11:55a
RBFILES      <DIR>     01-03-91   9:51p
SPRINT       <DIR>     01-03-91  10:20p
APPDIR   BAT        34 04-29-This Is The DOS Screen
AUTOEXEC CLR       160 04-20-91   7:25a
AUTOEXEC 50        427 04-15-91   8:33a
AUTOEXEC BAT       385 05-04-91   8:00p
AUTOEXEC ORG       326 01-04-91  11:56a
AUTOEXEC DV        385 05-04-91   8:00p
CONFIG   DV        210 03-02-91   1:14p
CONFIG   ORG       192 01-04-91  11:56a
CONFIG   50        236 04-17-91   7:22a
CONFIG   SYS       253 05-05-91  10:38p
CONFIG   CLR       221 04-20-91   7:44a
DV       BAT        55 01-04-91  11:58a
MIRROR   BAK     58880 05-05-91  10:33p
MACS     BAT        96 03-30-91  11:42a
MIRROR   FIL     58880 05-05-91  10:38p
OPT3     BAT       642 01-04-91  12:28p
HELP     BAT       603 05-05-91   6:03p
SETCL    BAT        62 04-20-91   7:00a
```

Figure 9-3 —ANSI-defined DOS Prompt without First Clearing Screen.

The following four escape sequences may be used with the PROMPT command to move the cursor up, down, left and right:

$E[numA	Moves the cursor up **num** rows without changing the column it is in.
$E[numB	Moves the cursor down **num** rows without changing the column it is in.
$E[numC	Moves the cursor to the right **num** spaces without changing the row it is in.
$E[numD	Moves the cursor to the left **num** spaces without changing the row it is in.

One use of the C and D sequences is to make detailed changes to the appearance of the date and time in the DOS prompt, as demonstrated in the following command:

```
PROMPT $D$E[10D$H$H$H$HDate$E[9C$G
```

This command replaces the day of the week with the word Date. The result is the following prompt:

```
Date 06-01-1991>_
```

The first parameter, **$D**, inserts the date into the prompt. The prompt is now "Sat 06-01-1991." The first escape sequence, **$E[10D**, moves the cursor to the left 10 spaces, which puts it under the first 0 at the beginning of the date. The following four parameters are all **$H**, which erases four characters (the day of the week and the space that separates it from the date) and leaves the cursor at the beginning of the line. Then the word "Date" is inserted, followed by the escape sequence **$E[9C** which moves the cursor to the right nine spaces, placing it at the end of the date. Finally there is a **$G** parameter, which causes the prompt to terminate with a greater than sign.

PROMPT Parameters Used for Modifying the Screen Display

You may use escape sequences to modify the screen display itself, as well as to position the cursor. The extent to which you can modify the display is dependent on the type of monitor you have. If you have a color monitor, you may set the foreground and background colors of characters that appear in the prompt, as well as the colors of anything that is displayed as the result of a command. This includes information such as directory displays. The escape sequence settings do not, however, affect the display of the shell or other programs you run — it only affects displays generated by DOS commands. On monochrome displays you can define the underline characters.

The escape sequence **$E[2J** is used to erase the entire screen and position the cursor in the upper left corner. You may then use the cursor positioning sequences to place it elsewhere if you desire. This would be a useful sequence to add to the example above that displays a message in the middle of the screen using the **$E[row,columnH** sequence.

The escape sequence **$E[K** erases the screen from the current cursor position to the end of the line. This erases whatever is at the cursor position as well, so before using this sequence, you may want to move the cursor to the right one space to avoid erasing the last character of the prompt.

The format of the escape sequences for setting colors and other display attributes is:

```
    $E[setting1;setting2;...m
```

There may be any number of settings; the last setting is terminated with a small m rather than a semi-colon. This indicates the end of the escape sequence. Each setting is a number between 0 and 47, though not all numbers are available. The available settings are as follows:

Setting	Description
0	Sets all other settings off. This sets all new displays the screen to the default of white letters on a black background. Information on the screen is unaffected.
1	Turns on bold. Everything displayed after this setting will appear in bold (high intensity). This will affect everything displayed until an "all settings off" (setting 0) turns bold off. Anything displayed in bold will remain in bold on the screen; new characters will not be in bold.
4	Turns on underscore. This is effective for monochrome displays only — color displays cannot show a character as underlined. This will remain in effect until an "all settings off" (setting 0) turns off underlining.
5	Turns on blinking. Everything displayed entered after this setting will blink. This will affect everything displayed entered until an "all settings off" (setting 0) turns blinking off. Anything displayed before the $E0m will continue to blink, but new displays will not blink.
7	Turns on reverse video. Everything displayed entered after this setting will appear in black letters on a white background. This setting will remain in effect until an "all settings off" (setting 0) turns off reverse video. Anything displayed in reverse video will remain in reverse video after it is turned off.
8	Turns off the display screen. Anything typed at the keyboard and anything that would normally appear on the screen will not appear. This is useful for entering passwords that you do not want to have appear on the screen when you enter them. This remains in effect until an "all settings off" (setting 0) resets the display mode.
30	Sets the foreground color to Black.

31	Sets the foreground color to Red.
32	Sets the foreground color to Green.
33	Sets the foreground color to Yellow.
34	Sets the foreground color to Blue.
35	Sets the foreground color to Magenta.
36	Sets the foreground color to Cyan.
37	Sets the foreground color to White.
40	Sets the background color to Black.
41	Sets the background color to Red.
42	Sets the background color to Green.
43	Sets the background color to Yellow.
44	Sets the background color to Blue.
45	Sets the background color to Magenta.
46	Sets the background color to Cyan.
47	Sets the background color to White.

You can also use escape sequences to set the screen “mode,” where mode refers to how the screen as a whole displays information. There are two classes of modes: text and graphics. Text modes display only text and limited figures, but the resolution is very high. Graphics modes can display very complex graphic images, but when displaying text characters the resolution is sometimes not as high.

Text modes divide the screen into either 40 characters per line and 25 lines per screen, or 80 characters per line and 25 lines per screen. Graphics modes divide the screen into much smaller units called pixels. The number of vertical and horizontal pixels varies with the type of monitor and video card in your computer. The various types of cards are reviewed in Chapter 1. Pixels can either be in color or black and white.

In addition to setting either graphics or text mode, there is one additional use of the Set Mode escape sequence: to turn on or off the line wrapping feature. By default, if you type past the end of the DOS command line, the cursor moves to the beginning of the next line and you can continue typing there. You may enter up to 128 characters on the DOS command line, so depending on the length of the prompt the line may wrap once or twice. If you do not want to allow line wrapping you may use the Set Mode escape sequence to cause the cursor to stop at the end of the line. Whatever you type at that point overwrites the last character on the line.

Which modes are available on your computer depends upon your hardware configuration: some monitors can only display in one or two modes, while others can display in all of them. All monitors will display in 40 X 25 and 80 X 25 text modes, and all monitors will allow the line wrapping feature to be switched on and off. If you aren't sure which modes are available on your computer, the simplest way to find out is to try out the commands and see which ones affect your display.

The mode is set with the escape sequence **$E[=numh**, where **num** is a number that specifies the mode and the **h** marks the end of the command. The h must be a lowercase h. The allowable numbers and their effect are specified in Table 9-1.

Table 9-1 — Screen Mode Settings Using the ANSI.SYS Driver.

Setting	Description
0	40 X 25 characters text mode - black and white
1	40 X 25 characters text mode - color
2	80 X 25 characters text mode - black and white
3	80 X 25 characters text mode - color
4	320 X 200 pixels graphics mode - 4 colors
5	320 X 200 pixels graphics mode - black and white
6	640 X 200 pixels graphics mode - black and white
7	turns on line wrapping (the default)
14	640 X 200 pixels graphics mode - 16 colors
15	640 X 350 pixels graphics mode - 2 colors
16	640 X 350 pixels graphics mode - 16 colors
17	640 X 480 pixels graphics mode - 2 colors
18	640 X 480 pixels graphics mode - 16 colors
19	320 X 200 pixels graphics mode - 256 colors

You may use a lowercase L in place of the lowercase h to terminate these escape sequences. The effect is the same, except in the case of setting 7. The sequence **$E[=7l** will turn off the line wrap feature, so that the cursor stays at the end of the line when you reach it.

PROMPT Parameters Used for Modifying the Keyboard

The **$E** escape sequences may be used for modifying the way the CPU responds to the keyboard. Specifically, you can assign any key (including shift-key combinations and function keys) to print a different letter than was originally assigned to it, or even to print a string of letters, numbers and symbols. For example, you could assign F1 to automatically perform a DOS command, such as DIR, which lists the files in the current directory. You may also use this feature to cause a message to display whenever a particular keystroke is entered.

These keyboard commands work only for keystrokes entered at the DOS prompt. They will not affect the keystrokes entered within the Shell or within your software programs, such as your word processor. They can therefore be used to customize the DOS interface without worry about interfering with other programs.

To understand how this works you must know a little about how the computer responds to the keyboard. Each key, and each ⇧Shift, alt and Ctrl key combination, produces a numeric code that gets sent to the CPU. These codes are translated by the computer into ASCII codes. ASCII codes are codes that correspond to the common characters and symbols that we use for communicating — the upper and lowercase alphabetic characters, numbers, punctuation symbols, and so on. When the CPU sees one of these ASCII codes it displays the appropriate character on the screen. For example, the ASCII code for capital A is 65, so when the CPU sees an ASCII code of 65 it displays a capital A on the screen.

There are 256 ASCII codes altogether. The first 128 of them (codes 0 through 127) are known as standard ASCII. They share common meaning among any computer system that uses ASCII codes, and they represent such common symbols as the letters of the alphabet or numbers.

The use of the last 128 codes (codes 128 through 255) is not shared among computer systems. IBM-compatible PCs use these codes for special symbols that can be displayed on the screen. They include letters of foreign alphabets, arrow symbols, and symbols that can be used for drawing lines and boxes on the screen. The entire 256 code ASCII table for IBM-compatible PCs appears in the appendices. Some of the ASCII codes produce two symbols — a caret (^) followed by a capital letter. Some of them do not produce printable characters at all — they are used for sending special messages to the display device (either a printer or the screen). An example of such a message is CR, for Carriage Return. This tells the printer or screen to move to the beginning of the next line.

Escape sequences can be used to change the ASCII code that is assigned to a key. The escape sequence used for this purpose is:

```
$E[num;replacementp
```

In this escape sequence **num** stands for the ASCII code for the key, **replacement** stands for the new code or codes to be assigned to that key, and **p** indicates the end of the sequence. The re-assignment codes may be either in the form of another numeric code or actual letters enclosed in double quotes. For example, the letter A (uppercase A) has an ASCII code of 65 and the letter B has an ASCII code of 66. Both of the following PROMPT commands will cause the letter B to appear when the letter A is entered:

```
PROMPT $E[65;66p
PROMPT $E[65;"B"p
```

Since these are PROMPT commands, as well as commands sent to ANSI.SYS, they will change the DOS prompt — in fact, they will eliminate it, since there are no PROMPT parameters other than $E in the command. To send this keyboard command to ANSI.SYS and also leave a DOS prompt that shows the current directory followed by a greater than sign, you must enter one of the following commands:

```
PROMPT $E[65;66p$P$G
PROMPT $E[65;"B"p$P$G
```

After either of these PROMPT commands has been entered, typing a capital A at the DOS prompt will result in a capital B appearing on the screen. You may assign more than ASCII code to a key. This allows you to have entire messages or commands appear at the touch of a key. These messages may be entered as either ASCII numeric codes or as character strings enclosed in quotes. Both of the following commands assign the word HELP to the A key:

```
PROMPT $E[65;72;69;76;80p$P$G
PROMPT $E[65;"HELP"p$P$G
```

Again, the PROMPT is terminated with PG to leave a reasonable prompt appearing on the screen.

You may assign codes to the function keys rather than the regular keys by starting the numbers in the escape sequence with a 0. This tells ANSI.SYS that the next number refers to what is known as an “extended” code. The function keys are numbered with extended codes 59 through 68. The following PROMPT re-assigns **F1** to type the message “HELP:”

```
PROMPT $P$G$E[0;59;"HELP"p
```

If you examine the ASCII table, you will see that ASCII code 13 has the letters CR corresponding to it. CR stands for Carriage Return, and whenever the CPU sees the ASCII code 13 it sends a carriage return signal to the output device. This is the same as having pressed Enter↵. An ASCII code 13 can be included in the keyboard re-assignment escape sequence to cause a keystroke to carry out a DOS command. For example, the following PROMPT command results in the letters DIR, followed by a carriage return, to be typed out whenever F1 is pressed:

```
PROMPT $P$G$E[0;59;"DIR;"13p
```

This will cause the DOS DIR (directory) command to be entered when F1 is pressed, resulting in a directory listing of the current directory. You can extend this to have function keys execute batch files.

A particularly useful application of this technique is to use it with an appropriate DOS prompt to make sure you or other users of your computer always know how to get back to the Shell. The following PROMPT command will set this up:

```
PROMPT Press F1 to return to the Shell$G$E[0;59;"EXIT;"13p
```

This will result in the following DOS prompt:

```
Press F1 to return to the Shell>
```

It will also assign the word EXIT, followed by a carriage return, to F1, which is what is required for returning to the Shell when you have temporarily exited from the Shell. For this prompt to remain in effect when you exit and re-enter the Shell you must enter the PROMPT command at the DOS prompt (while the Shell is not running in the background).

Fine-tuning the Keyboard Response Rate with the MODE Command

The MODE command can be used to adjust the rate at which the computer repeats a key that is held down, and the amount of time it waits after the key is pressed before beginning to repeat. The first of these two characteristics is called the "typematic rate," and the second is called the "keyboard delay." However, some keyboards, particularly older ones, will not recognize this command.

To adjust the rate, enter the following command:

```
MODE CON RATE=xx DELAY=yy
```

The value of **xx** may be anywhere from 1 to 32, representing about 2 to 30 repetitions per second. The default value is 20 or 21, depending on the keyboard.

The value of **yy** may be 1, 2, 3, or 4. These correspond to 0.25, 0.50, 0.75, and 1.00 seconds, respectively. The default value is 2 (0.50 seconds).

You must either set both values, or neither. You cannot set just one of them.

Managing the Print Queue

When you use the PRINT utility, all files that are sent to the printer are put into the print queue. The print queue is an area of memory, which stores the names of files to be printed. When one file finishes printing the next one in the queue is sent to the printer.

The PRINT utility is enabled by including a PRINT command in the AUTOEXEC.BAT file. You may modify the PRINT command in the AUTOEXEC.BAT file using the switches discussed below. Many of the switches can only be executed the first time the PRINT command is invoked. These should be set in the AUTOEXEC.BAT file, since that is where the PRINT command is first called. These may only be set once because the first time PRINT is executed a portion of the program is loaded into memory, and these switches are set in that portion of the program. They cannot be manipulated after being loaded.

The other switches are used for manipulating the queue after it has been called. They allow you to remove files from the queue and add files to the queue, and they can be invoked as often as you wish. These are available by going to the command prompt and executing the PRINT command.

You may use as many or as few of the switches as you like. If you do include switches the first time you invoke PRINT, DOS will prompt you for the device to print to (discussed in the next section, "Specifying the Printing Device"). If you invoke PRINT without any switches after the first time it has been invoked, it will simply list the files currently in the print queue.

If you want the flexibility of manipulating the print queue from within the shell, you can add a PRINT QUEUE option to one of the menus (the DOS Utilities menu would be a logical choice). This option can then be used to set up and manipulate the print queue after you have placed files into it with the Print option on the File menu.

If you want control over the PRINT utility startup switches, you should remove the PRINT command from your AUTOEXEC.BAT file. You can then load the resident part of PRINT the first time you execute the PRINT command. You may specify the load switches at this time.

After a file has been added to the print queue, you must not make any changes to it until after it has finished printing. If the file is on a floppy disk you must leave the disk in the drive until the file has been printed. If a disk error occurs when DOS attempts to read a file for printing, the following sequence of events occurs:

1. The file being printed is canceled from the queue.
2. The disk error message is printed on the printer.
3. The printer paper is advanced to the next page.
4. The printer alarm is sounded.
5. The remaining files in the print queue are printed.

You must not attempt to use the printer for any other purpose while the Print command is operating. For example, you must not press the ⇧Shift-print screen combination to print a screen image. Doing so will cause the message "out-of-paper" to appear until all files have been printed or printing is terminated with a /C or /T switch (discussed below).

Specifying the Printing Device

Devices such as printers are attached to the computer through connections called ports. The print program must know which port should receive information that is to be printed. For instance, you may have two printers attached to your computer at ports named LPT1 and LPT2 (for Line Printer 1 and Line Printer 2). When you run PRINT the first time, you may specify where the information should be sent. This is done with a /D switch. The /D switch can only be entered the first time you run the PRINT command. If you use /D, it must be the first switch specified. The following command installs the resident part of the print program and tells it to send all files in the print queue to the LPT1 port:

```
PRINT /D:LPT1
```

If you do not include a /D switch, the first time you invoke PRINT the computer will respond with the message:

```
Name of list device [PRN]:
```

Pressing Enter← will accept the default value of PRN, which in most cases is the correct choice. If you have one printer attached to your computer on the LPT1 port, this default value will work. Otherwise, you may specify any other port to which you would like to send the print queue output. Common choices are shown in Table 9-2.

Table 9-2 — Common Ports for Printer Connections.

LPT1	line printer 1
LPT2	line printer 2
LPT3	line printer 3
PRN	same as line printer 1
COM1	communications port 1
COM2	communications port 2
AUX	auxiliary port

You must make sure that the port identified by the /D switch exists on your computer, and that there is some device attached to it that can receive the output. If there is no such port, or if there is nothing attached to it, using the PRINT command may cause your computer to hang up or to work erratically. For example, if you specify LPT2 your computer must have a port named LPT2 and there must be a printer attached to it.

Specifying the Size of the Print Buffer

When you run Print the first time after booting your computer, it sets up an area of memory known as a print buffer. Many printers can only receive a small number of characters at a time, and after they have printed these characters they request more from the computer. The computer must then suspend whatever it is doing, read the next characters to be printed from the disk, and send them to the printer. This ends up causing large portions of the computer's time to be spent communicating with the printer.

The print buffer is used to alleviate this problem. It stores a large number of characters that are ready to go to the printer, so when the printer is ready for more characters the computer does not need to suspend other operations in order to read them from disk. The only time the computer needs to suspend other operations for the printer is when the print buffer is empty.

The /B switch is used to specify the number of bytes set aside for the print buffer. It can only be entered the first time you invoke the PRINT command. If you do not include the /B switch, a 512-byte buffer is used. You may specify anywhere from 512 bytes up to 16K bytes. A larger value may noticeably increase the performance of the PRINT command, especially if you have a high-speed printer.

The following command demonstrates the /B switch:

```
PRINT /B:1024
```

This specifies a print buffer of 1024 bytes — twice the default value of 512. Since /B can only be used the first time you invoke PRINT you will probably include a /D switch, as follows:

```
PRINT /D:LPT1 /B:1024
```

Remember that the /D switch must be the first switch if it is to be included; the order in which the other switches appear is not important.

Specifying the Maximum Number of Files in the Queue

The /Q switch is used to specify the maximum number of files you may have in the queue at any one time. If you reach this maximum DOS will not allow you to add any more until the first one finishes printing. Then you may add one more, until the next one finishes, and so on.

The /Q switch can only be added the first time PRINT is invoked. You may specify anywhere from 4 to 32 files. If /Q is not included, the default value is 10. The following command demonstrates the use of /Q:

```
PRINT /D:LPT1 /B:1024 /Q:15
```

This command specifies LPT1 as the printer port, 1024 bytes for the print buffer, and a maximum of 15 files in the queue at any one time.

Managing PRINT Time

When the Print queue is in use, it slows down the operation of your computer. There is no way around this. The computer can only do one thing at a time, so when it tends to the print queue it cannot tend to your other tasks. It is important, therefore, to have some control over how much of the computer's time is devoted to managing the print queue and how much of it is available for your other tasks.

There are three factors that determine how much of the computer's time is spent managing the Print queue. But to understand these you must understand a little bit about how the computer's clock controls what the computer does.

Like any clock, the computer's clock operates by "ticking" time away. It ticks very fast — millions of clock ticks per second. Since a tick is such a tiny amount of time, the computer allocates much of its time in terms of "time slices" rather than clock ticks: a time slice consists of 255 clock ticks, just as a minute consists of 60 seconds.

The computer operates in "time cycles." People also operate in time cycles, called days. A person may get up at 6:30; have breakfast at 7:00, arrive at work at 7:45, leave work at 4:45, and so on. Each day repeats the cycle. Certain time slices of the day are allocated to particular tasks, such as having breakfast. In the same way, the computer breaks its cycle into different tasks.

If you are working in your office on three projects, you may allocate the time from 1:00 to 3:00 for a particular one. Each day you work on that project from 1:00 to 3:00, and each day a little more of it gets done. In the computer's cycle, each time it gets around to the point where a particular task gets some time, a little more of that task gets done.

It is important to keep in mind that in the computer all of this is happening very fast — thousands of times per second — so you rarely notice the computer pausing in one task to give time to another.

In your computer's cycle there are 255 slices available for all tasks. This means that the smallest amount of time the computer can allocate to any one task is 1/255th of its cycle. You can specify how many time slices per cycle get allocated to managing the print queue with the /S switch. This switch can only be entered the first time you run the PRINT command. You may specify anywhere from 1 to 255 time slices. If you do not specify the /S switch the default value of 8 is used. The following command demonstrates the use of the /S switch:

```
PRINT /D:LPT1 /S:16
```

This command specifies the LPT1 port, and 16 time slices per cycle — (double the default value). The effect of this is that more of the computer's time will go to making sure your files print as fast as possible, but less time will be available to whatever other task you are performing while the files are printing.

When you specify a number of time slices (16 in this example), the computer does not bunch all of them together (give the printer 16/255ths of its time all at once, then wait for a complete cycle to come around before the PRINT program gets any time again). Rather, it distributes the slices evenly throughout the cycle. This helps make the print operations more transparent to you, since only very small amounts of time are being taken away from your other tasks. But those small amounts are taken away more frequently.

When a time slice for the print queue comes around, DOS checks to see if the printer is still printing the last characters sent to it or if it is waiting for some more characters. If it is waiting for more, DOS starts sending them.

However, DOS can send characters to the printer much faster than the printer can print them, so much of the time slice allocated to printing may be wasted. DOS may be able to send all of the characters the printer can print between now and the next printer time slice in the first few clock ticks of this slice, so the rest of the clock ticks for this slice are wasted. The /M switch is used to recover those ticks for the next task in the computer's cycle.

The /M switch specifies the number of clock ticks that DOS will use to send characters to the printer. If the printer is waiting for characters when a printer time slice starts, only the number of ticks specified with /M will be used to send characters to the printer. The rest of the time slice is released for the next task in the computer cycle.

You only specify /M the first time the PRINT command is invoked. You may specify anywhere from 1 through 255 ticks. The default value used if /M is not included is 2. The following command demonstrates the use of /M:

```
PRINT /D:LPT1 /M:10
```

This command specifies the LPT1 port, and allocates 10 clock ticks per printer time slice for sending characters to the printer. Since no /S switch is included in the command the default of 8 time slices per cycle is used.

If the printer is still printing when a printer time slice comes around, DOS will continue checking the printer during the time slice. This can end up wasting the entire time slice if the printer does not complete its printing task during that time. The /U switch prevents this. It specifies how many clock ticks DOS will wait for the printer to complete its printing so it can receive new characters. If the printer does not finish in the number of ticks specified the rest of the time slice is released for the next task in the computer's cycle.

The /U switch can only be specified the first time Print is invoked. You may specify anywhere from 1 to 255; the default value if /U is not specified is 1. The following command demonstrates the use of the /U switch:

```
PRINT /D:LPT1 /U:10
```

This command specifies the LPT1 port, and 10 clock ticks to wait for the printer to be freed up. Since no /S or /M switches were specified, the default values of 8 and 2 will be used for them.

The best use of the /S, /M, and /U switches is totally dependent on both your hardware configuration and the use you will make of the Print utility. In most cases, the default values are satisfactory. However, if you find that the computer is taking too long to print your files, or that it is taking too much time away from your other tasks, you may want to try adjusting them.

The switch that is most likely to have an impact on how much the Print utility is slowing down your other tasks is the /S switch, since that determines how much total time out of each cycle gets devoted to Print. The /M and /U are for fine-tuning — they determine how much of the time slices allocated to the printer can be wasted before releasing the rest of the slice for other tasks.

If your printer is pausing frequently and for long periods of time, try increasing /M first. This will provide a larger portion of the available time slice for sending characters to the printer without taking up more time slices for printing. If that does not work increase /S. You may need to try several combinations of /M and /S before finding the one that provides the best printing speed with the least degradation of other computer performance.

The /U switch should only be used if maximizing printer throughput is very important, since it almost guarantees that time will be wasted waiting for the printer to become ready for more characters.

Managing Files in the Queue

There are three switches used to manipulate the files in the queue. These switches may be used at any time — they do not have to be included in the PRINT command the first time it is invoked.

The /T switch terminates printing from the queue. When you enter the PRINT command with the /T switch, all files in the queue are removed from the queue. The following command demonstrates the use of the /T switch:

```
PRINT /T
```

If a file is printing when the /T command is entered, the following sequence of events occurs:

1. The file being printed is canceled from the queue.
2. A file cancellation message is printed on the printer.
3. The printer alarm sounds.
4. The printer paper is advanced to the next page.
5. Any files specified on the command line after the /T switch are printed.

If no files are in the queue when this command is entered the message "PRINT queue is empty" will appear on the screen.

You may also specify one or more files to print after emptying the queue. These are included on the command line after the /T switch. The reason for canceling the print queue is often because you want to print a file immediately, without waiting for all the other files in the queue to complete. The following command demonstrates this:

```
PRINT /T LETTER1.FIL LETTER2.FIL
```

This command will cancel all files in the queue and put the files LETTER1.FIL and LETTER2.FIL into the queue. LETTER 1.FIL will then print, followed by LETTER2.FIL.

The /C switch is used to cancel specified files from the queue. You may include one file name before the /C and as many as you like following it. Both of the following commands will remove the files LETTER4.FIL and LETTER6.FIL from the print queue:

```
PRINT LETTER4.FIL /C LETTER6.FIL
PRINT /C LETTER4.FIL LETTER6.FIL
```

You do not have to specify the file names in the order that they occur in the queue. If /C is used to cancel a file that is currently being printed the following sequence of events occurs:

1. The file being printed is canceled from the queue.
2. A file cancellation message is printed on the printer.
3. The printer alarm sounds.
4. The printer paper is advanced to the next page.
5. The remaining files in the print queue are printed.

The /P switch adds files to the print queue. The file name preceding the /P and all file names following it will be added to the print queue. Files are added in the order in which they occur on the command line. Both of the following commands will add the files LETTER4.FIL and LETTER6.FIL to the queue:

```
PRINT LETTER4.FIL /P LETTER6.FIL
PRINT /P LETTER4.FIL LETTER6.FIL
```

The /P switch is the default: if no switches are specified, but file names are included on the command line, then the files are added to the print queue. The following two command lines are functionally identical:

```
PRINT LETTER4.FIL /P LETTER6.FIL
PRINT LETTER4.FIL LETTER6.FIL
```

The /C and /P switches may be combined on one command line. The switch specified first will be in effect up to the file name that precedes the switch specified second. Either of the following command lines will remove the file LETTER4.FIL from the queue, and add the files LETTER5.FIL and LETTER6.FIL:

```
PRINT LETTER4.FIL /C LETTER5.FIL /P LETTER6.FIL
PRINT LETTER4.FIL /C /P LETTER5.FIL LETTER6.FIL
```

To experiment with the commands, you can turn off your printer and enter the PRINT command with various switches. If you turn off your printer the print queue will not empty out. Whenever any Print command executes, the entire queue is listed, so you can see what effect the command had.

Use of Wild Cards in Print Command File Names

You may use the wild cards * and ? in the file names entered in the PRINT command. The * is used to replace groups of characters, the ? is used to replace individual characters. For example, the file name *.FIL refers to all files that have an extension of FIL. The file name LETTER?.* refers to all files that begin with LETTER plus one more character, and any extension. This would include LETTER1.FIL, LETTER2.DAT, etc. You may also embed the asterisk within a file name: LET*.FIL will include LETTER.FIL, LET.FIL, LETTUCE.FIL, etc. However, if you begin the file name with * all file names will be included. The designation *ETTER.FIL will include all files with the extension FIL.

If you wanted to print all files with an extension of FIL you could enter the following command:

```
PRINT *.FIL
```

To print all files that begin with letter followed by a number and the extension FIL you could enter the following command:

```
PRINT LETTER?.FIL
```

You could also print all files that begin with let followed by any group of characters and the extension FIL with the following command:

```
PRINT LET*.FIL
```

The wild card symbols may be included with any of the PRINT switches. The following command will remove all files that are named letter followed by an extension:

```
PRINT LETTER.*/C
```

Specifying the Maximum Number of Logical Drives

A logical drive is any device or structure that appears to DOS as a disk drive. Thus, setting up a RAM drive creates a logical drive, and partitioning a single hard disk into several smaller drives creates multiple logical drives. A logical drive is anything referenced by its own drive letter.

You may have to up 26 logical drives on your computer — one for each letter of the alphabet. But for each drive letter that is supported, a small amount of memory is required. By limiting the number of drives that can be supported, you can conserve on memory. The LASTDRIVE command is used to do this. LASTDRIVE is a CONFIG.SYS command, and is used to define the last drive letter that DOS will recognize. Normally, if you have a PC with one or two floppy drives and a single hard drive, the last drive letter is the one for your hard drive — C. You could therefore include the following command in your CONFIG.SYS file:

```
LASTDRIVE=C
```

The only reason for providing extra drive letters is if you expect to create a new logical drive with the SUBST command. SUBST is discussed in detail in Chapter 6. SUBST is used to assign a logical drive letter to replace a directory path. These substituted drive letters are sometimes called virtual drives. If you want to do this, you will need to adjust the LASTDRIVE value to leave enough free drive letters for as many virtual drives as you want to create.

Controlling the Effects of Control-Break

Pressing Ctrl-C, or Ctrl-break, interrupts the activity of the CPU and terminates whatever process is taking place. For example, if a very long directory listing is in progress, you can interrupt it by pressing Ctrl-C. This is because DOS scans the keyboard for these keystrokes. However, normally DOS only watches for Ctrl-C when it is reading keyboard input or writing something to the screen or printer. During other operations, like disk reads and writes, it does not check for Ctrl-break. This makes sense, since these are operations that should not normally be interrupted. However, you can tell DOS to scan for Ctrl-break during all DOS operations by entering the following command at the command prompt:

```
BREAK ON
```

You may turn this extended checking off again by entering this command:

```
BREAK OFF
```

By default, BREAK is off. Turning it on slows down the performance of your computer slightly, because DOS is scanning for Ctrl-break more frequently.

The effect on your software programs of turning on Ctrl-break is not predictable. On some, it may permit you to interrupt some processes that would otherwise not be interruptible, while for other programs there will be no effect.

Chapter 10

The DOS Command Dictionary

This chapter provides a command dictionary of all commonly used DOS commands. A statement of each command's purpose and its syntax are given. Some of these commands are discussed in greater detail in other chapters of this book. You may locate these discussions by checking the index.

A type is also specified for each command, and it will be one of the following:

- Internal
- External
- External/Internal
- CONFIG.SYS
- Batch.

Internal commands are loaded into memory when you boot your computer. It is not necessary to have a DOS disk or directory available when you invoke these commands, since they are located in memory.

External commands are not loaded into memory when the computer is booted. Consequently, you must have a DOS disk available on a floppy disk system, or have the DOS directory available (via a PATH or APPEND if it is not the current directory) in order to use the command.

External/Internal commands are external the first time they are invoked. The DOS disk or directory must therefore be available when they are used for the first time. After their first use they become internal commands — they are then retained in memory and can be used without access to a disk. When you reboot your computer they become external until they are invoked again.

CONFIG.SYS commands must be specified in the CONFIG.SYS file. In general, these commands cannot be entered at the command prompt, though there are some exceptions to this. If they can be entered at the command prompt as well, they will have two labels — Internal and CONFIG.SYS.

Batch commands are used either exclusively or primarily in batch files. These commands are generally usable in Shell programs as well.

@

Type:	Internal.
Purpose:	Used in batch files to suppress the echoing of a command, even if ECHO is set to ON. Has no effect if ECHO is set to OFF.
Syntax:	`@ command`
Parameters:	**command** is any legal batch file command.
Comments:	Useful if you do not want to set ECHO OFF, but do want to suppress the display of a batch file command.

APPEND

Type: External/Internal.

Purpose: Sets a path of directories that DOS will search when it cannot locate a file in the current directory. The /X parameter can be specified to tell DOS whether or not to use the APPENDed directories when searching for executable files (files with a COM, EXE, or BAT extension). The APPENDed directories will always be used when searching for non-executable files. The `/PATH` parameter may be used to tell DOS whether to search APPENDed directories for files that included a drive or path specification.

Syntax:

```
APPEND path1; path2 ... /X:OFF /PATH:OFF
                           ON        ON

APPEND /E /X:OFF /PATH:OFF
             ON        ON

APPEND ;

APPEND
```

Parameters: `/X` same as /X:ON.

`/X:ON` turns on the use of APPENDed directories when searching for executable files. `/X:OFF` turns off the use of APPENDed directories when searching for executable files. The default is `/X:OFF`.

`/PATH:ON` tells DOS to search APPENDed directories even if a drive and/or path is specified with the file name. The default is `/PATH:ON`. `/PATH:OFF` tells DOS not to search APPENDed directories when a drive and/or path is specified with the file name.

`/E` tells DOS to keep the APPEND paths in the DOS environment. This permits the APPEND paths to be viewed with the `SET` command as well as with the `APPEND` command. /E can only be used the first time `APPEND` is invoked. If /E is used then you may not specify any paths to be APPENDed the first time it is invoked, you may specify paths to be APPENDed only on subsequent calls to `APPEND`.

`;` The semicolon is used to separate paths that are being APPENDed. If the APPEND command is entered with just the semicolon the previous APPEND command is canceled.

Comments: Each time you enter a new APPEND command the preceding one is canceled. Consequently, only the most recent APPEND command is in effect at any time. If APPEND is entered with no parameters the first time, it is loaded into memory and becomes an internal command. If APPEND is entered with no parameters after the first time, a list of APPENDed paths is displayed.

If a file is found in an APPENDed path by a program and is then saved, it will be saved in the current directory. This can lead to multiple versions of the same file existing in different directories.

If the ASSIGN command is to be used, as well as the APPEND command, the first APPEND command must be entered before the ASSIGN command

If a path that does not exist is included in a list of APPEND paths, DOS will ignore that path and go on to the next. However, if a path cannot be found because the drive on which the path exists is not ready, then DOS terminates the APPEND search.

ASSIGN

Type: External.

Purpose: Causes a reference to a drive to access a different drive.

Syntax: `ASSIGN drive1=drive2 drive3=drive4 ...`

Parameters: `drive1` and `drive3` refer to drives being reassigned

`drive2` and `drive4` refer to drives that `drive1` and `drive3` are being reassigned to.

Comments: You should not use the JOIN or SUBST commands if ASSIGN is in effect. Some programs will ignore the effect of ASSIGN and use the drive specified even if it has been re-assigned.

ASSIGN commands are cumulative. If you assign drive A to drive B, then later assign drive C to drive, drive A will continue to be assigned to drive B. You can reset all drive assignments to their normal meaning by entering the command ASSIGN with no parameters.

The SUBST command provides the same functionality as ASSIGN, with more flexibility. ASSIGN may disappear in future versions of DOS. It is therefore recommended that you use SUBST instead.

ATTRIB

Type: External.

Purpose: Used to switch the Read-only, Archive, System, and Hidden attributes on and off. Can be used on one or multiple files using * and ? wild cards. Can also be used to display a file's attribute settings.

Syntax:

```
ATTRIB +/-R +/-A +/-H +/-S filespec /S
ATTRIB filespec
```

Parameters:

`+R`	turns on the read-only attribute
`-R`	turns off the read-only attribute
`+A`	turns on the archive attribute
`-A`	turns off the archive attribute
`+S`	turns on the system attribute
`-S`	turns off the system attribute
`+H`	turns on the hidden attribute
`-H`	turns off the hidden attribute

`filespec` specifies the file or files whose attributes should be set. Wild card symbols * and ? can be used to change attributes in sets of files. Drive and path specifiers may be included.

`/S` causes all files in a specified directory and its subdirectories to be affected by the settings. For example, `ATTRIB +A C:\WP\*.* /S` will turn on the archive attribute for all files in the directory WP and all of its subdirectories.

Comments: When the only parameter is a file name, DOS displays the attribute settings for that file. For example, entering `ATTRIB MYFILE.LET` will result in a message similar to: `R C:\MYFILE.LET` This indicates that MYFILE.LET has the read-only attribute set on and the other attributes set off.

If you attempt to delete a file that has the R, H, or S attribute set, DOS will display the message "Access denied." You may use the `ATTRIB` command to turn off these attributes.

The /S parameter may be used to locate all occurrences of a file on a disk. For example, the command `ATTRIB C:\MYFILE.LET /S` will cause DOS to display all occurrences of the file MYFILE.LET on drive C.

BACKUP

Type: External.

Purpose: Use to back up files from one disk to another disk. The files may be restored with the RESTORE command.

Syntax:

```
BACKUP source-filespec target-drive /S /M /A /F:size
/D:mm-dd-yy /T:hh:mm:ss /L:filename
```

Parameters: `source-filespec` specifies the file(s) to be backed up; a drive and path may be included. If only a drive is specified, all files in the current directory of that drive will be backed up. If a drive and path are specified, all files in the specified directory will be backed up. If a file name is specified, only the file(s) specified by the file name will be backed up. The handling of subdirectories is determined by the /S parameter.

`/S` backs up all files that match the file name specification in the current directory and its subdirectories.

`/M` backs up those files that have been modified since the last backup. This is determined by checking the ARCHIVE attribute of the file(s).

`/A` adds the files being backed up to files that already exist on the target disk. If /A is not specified all existing files on the target disk will be erased by the BACKUP command.

`/F:size` formats the disk to the specified size. Possible values of size are: 160KB, 180KB, 320KB, 360KB, 720KB, 1200KB, 1440KB, 2880KB.

`/D:mm-dd-yy` backs up files that were modified on or after the date specified. The format of the date depends on the Country setting. This setting is determined during the installation of DOS 5.0. It may be modified with the COUNTRY command.

`/T:hh:mm:ss` backs up files that were changed at or after the time specified on the date indicated with the /D parameter. If /D is not used with /T then all files changed at or after the time specified on any date are backed up. Whether the time is interpreted as a 12-hour format or 24-hour format is dependent on the Country setting. This setting is determined during the installation of DOS 5.0. It may be modified with the COUNTRY command.

`/L:filename` creates a log file. If the /L parameter is used without a file name, the name BACKUP.LOG is used. If the specified log file already exists the new information is appended to it. The first line in the log file indicates the date and time of the backup. Subsequent lines contain the backup disk number, path, and file name of each back-up file.

omments: BACKUP will automatically format a floppy disk if it is not already formatted. For it to do this, the FORMAT program must be in the current directory or in a directory specified by the path. If the FORMAT program cannot be located the BACKUP command will display an error message and terminate the backup process.

Files can be backed up from:

1. A fixed disk to a floppy disk
2. One floppy disk to another floppy disk
3. A floppy disk to a fixed disk
4. One fixed disk to another fixed disk.

When one floppy disk is filled up, BACKUP automatically prompts you to put in the next disk. The disks are numbered by BACKUP to indicate the order in which they occur.

Files created with the BACKUP command are not ordinary files. They cannot be accessed directly by DOS or by your programs. You must use the RESTORE command to make them accessible.

The BACKUP command sets an Exit code that can be accessed by th Batch command `IF ERRORLEVEL ...` The Exit code is set accordin to the following table:

Exit code	Meaning
0	The BACKUP completed normally.
1	There were no files found to back up.
2	Some of the specified files were not backed up due to file-sharing conflicts on a network.
3	The BACKUP operation was terminated by the user pressing Ctrl-break.
4	The BACKUP operation terminated due to an error.

The DOS Utilities group that comes with the Shell includes Backup an Restore options that use these two DOS commands.

BREAK

Type: Internal and CONFIG.SYS

Purpose: Tells DOS whether to check for a Ctrl-break key stroke whenever a progran requests any DOS function.

Syntax: At the command line:

```
BREAK ON
```

```
BREAK OFF
```

In the CONFIG.SYS file:

```
BREAK=ON
```

```
BREAK=OFF
```

Parameters: `ON` tells DOS that whenever a DOS function is requested from an application program, DOS should check to see if the user has pressed Ctrl-break. If the user has pressed Ctrl-break DOS will terminate the program and return the user to the DOS prompt.

`OFF` tells DOS to check for a Ctrl-break combination only when one of the following operations is occurring:

1. Standard input operations
2. Standard output operations
3. Standard print device operations
4. Standard auxiliary device operations.

OFF is the default condition.

Comments: Setting BREAK on allows you to terminate some programs that use few or no standard device operations.

You may type `BREAK` at the command prompt to see the current BREAK setting

BUFFERS

Type: CONFIG.SYS

Purpose: Specifies the number of internal disk buffers that DOS should allocate.

Syntax: `BUFFERS=num1,num2/X`

Parameters: `num1` specifies the number of disk buffers to be used; the minimum is 1 and the maximum is 99 unless the /X parameter is used.

`num2` specifies the number of look-ahead buffers to be used; the minimum is 1 and the maximum is 8.

`/X` indicates that expanded memory is available and should be used for the disk buffers.

Comments: The default number of buffers is set to the greatest number determine according to the following specifications:

- If a floppy disk drive greater than 360K is installed the defaul number of buffers is 3
- If there is more than 128K of memory the default number o buffers is 5
- If there is more than 256K of memory the default number o buffers is 10
- If there is more than 512K of memory the default number o buffers is 15
- If none of these conditions are met the default number of buffer is 2.

Look-ahead buffers are used to read into memory not only the disk secto that contains the requested information, but also the specified number o sectors that follow that sector on the disk. If an application reads many data elements that are in consecutive order on the disk, this can enhanc disk performance. If the look-ahead buffers parameter is not specified non are used.

Each regular buffer increases the amount of memory required for DO by 532 bytes. Each look-ahead buffer increases the amount of memor by 512 bytes.

CALL

Type: Internal (only used in Batch files and Shell programs).

Purpose: Used in a batch file to execute another batch file and have control return to the first batch file when the second is finished executing.

Syntax: `CALL filespec`

Parameters: Drive and path specifiers may be included in the filespec. They are no necessary if the batch file being called is in the current directory or directory specified in the current path.

omments: Shell programs are passed to the DOS batch processor after their parameter dialogue boxes have been processed. Therefore, they appear to DOS as batch files, so you may use the CALL command to execute a batch file from a Shell program, then return control to the Shell program. If you execute a batch file from a Shell program without using the CALL command, control will not return to the Shell program.

When one batch file calls another, the first may pass parameters to the second. For example, if the line `CALL BATCH2 %2 %4` appeared in a batch file, the `%2` parameter of the first file would become the `%1` parameter in the second file, and the `%4` parameter in the first file would become the `%2` parameter in the second.

The ECHO mode is passed to the CALLed batch file. If that batch file changes the ECHO mode, it is restored when control returns to the first. Thus, if the first file sets ECHO to off and the second file sets ECHO to on, when the second file is called ECHO is initially off, and when control returns to the first file ECHO is reset to off even though the second file set ECHO to on.

Two or more batch files may repetitively CALL each other, or one batch file may CALL itself. However, it is essential that the cycle eventually ends, and that control is returned to the first batch file so it may finish executing.

If a batch file has the same name as an internal command (such as COPY), then attempting to CALL the COPY batch file will result instead in the internal COPY command executing. It is recommended that batch files be given unique names.

If you have an established set of batch files that you would like to execute from within the Shell it can be useful to add a generic CALL option to a menu.

CD or CHDIR

Type: Internal.

Purpose: Changes or displays the current directory of the specified drive.

Syntax:

```
CHDIR drive:path

CD drive:path

CHDIR drive:

CD drive:
```

Parameters: `drive` specifies the drive for which the current directory should b changed. If no drive is specified, the current drive is used. When only drive is specified, the current directory for that drive is displayed withou being changed.

`path` is the path to which the current directory for the specified driv should be changed. Path may not be more than 63 characters long, startin from the root directory.

Comments: Each drive has its own current directory. Thus, the commands `CD C:\DOS` and `CD A:\WP` will cause the current directory of drive C to be C:\DOS and the current directory of drive to be A:\WP. When you change drives the current directory is the one you are in.

If your prompt does not show the current directory, the CD command ca be useful for determining the current directory.

A leading backslash tells DOS to start at the root directory.

CHKDSK

Type: External.

Purpose: Displays the current status of the specified directory, the total amount c memory on the computer, and the amount of memory that is available fo programs.

Syntax:

```
CHKDSK drive:path\filename /F /V
```

where drive indicates the drive to be checked

Parameters: `path\filename` tells DOS to give additional report about the specified files (wild cards may be used to specify more than one file). For example, if a file named MYFILE.LET existed in the WP directory of drive C, the command `CHKDSK C:\WP\MYFILE.LET` would result in a report similar to the following:

```
Volume Serial Number is 2X5B-32BC
21309440 bytes total disk space
133120 bytes in 10 hidden files
55296 bytes in 20 directories
11771904 bytes in 641 user files
9349120 bytes available on disk
2048 bytes in each allocation unit
10405 total allocation units on disk
4565 available allocation units on disk
655360 total bytes memory
302656 bytes free
C:\WP\MYFILE.LET Contains 3 non-contiguous blocks
```

`/F` tells DOS to correct any errors it finds in the directory and file allocation table and to write the corrections on the disk. If you do not specify /F, DOS will go through the motions of correcting the errors, which gives you an indication of what effects the correction might have without actually writing the correction to the disk. If CHKDSK reports that there are "lost allocation units" or "lost cluster" on the disk, the /F parameter will cause CHKDSK to ask you if you want to recover them. Lost allocation units are sections of files that have become "orphaned" on a disk — DOS can no longer identify what file they belong to. Consequently the space they occupy is no longer available for other purposes. There are several possible causes of lost allocation units, but a power failure during operation is a common one. If you respond with a Y (Yes) to indicate that you want DOS to recover them, it will convert them to ASCII text files with names FILEnnn.CHK, where **nnn** represents a number. You can then examine them to determine whether the data in them is valuable.

`/V` tells DOS to display the names of all files on the specified drive. The path name for each files is displayed as well as the file name. DOS does not pause when the screen fills up, but pressing Ctrl-S will stop the screen from scrolling until you press another key.

Comments: The disk status information includes the total disk space; the number o hidden files on the disk and the number of bytes they occupy; the numbe of directories on the disk and the number of bytes they occupy; the numbe of user files on the disk and the number of bytes they occupy; the numbe of bad sectors on the disk; the number of bytes of available space on th disk; the size of the allocation units on the disk; the total number o allocation units on the disk; and the number of available allocation unit on the disk.

When you use CHKDSK to check a floppy drive, DOS assumes the driv is ready, so you must be sure to have a disk in the drive before executin this command.

It is a good idea to run CHKDSK periodically against any disks that ar in regular use. This will alert you to problems that may be developing i your files before you have lost a lot of valuable information.

If a file is badly fragmented (if it contains a large number of non-contiguous blocks), it can cause the computer to take longer than necessar to access the file.

CLS

Type: Internal.

Purpose: Clears the screen.

Syntax: `CLS`

Parameters: None.

Comments: This command is used to clear the screen. After the command executes the cursor and prompt appear at the top left corner of the screen. It is usefu in batch files to avoid having clutter obscure messages generated by th batch file.

COMP

Type: External.

Purpose: Compares the contents of one set of files to the contents of a second se of files.

Syntax: `COMP filespec1 filespec2 /D /A /L /N=num /C`

Parameters: `filespec1` specifies the first set of files to compare. It may include wild cards and a drive and path specification. If you specify a drive and path with no file name DOS assumes a file name of *.*.

`filespec2` specifies the second set of files against which the first is compared. It may include wild cards and a drive and path specification. If you specify a drive and path with no file name DOS assumes a file name of *.*.

`/D` Normally, COMP displays the differences between the two files in hexadecimal numbers. /D instructs COMP to use decimal numbers.

`/A` Displays differences as characters.

`/L` Displays the line number on which a mismatch occurs, instead of displaying the location or character.

`/N=num` compares the first **num** lines of both files, even if the files are of different sizes.

`/C` turns off case-sensitivity for the comparison.

Comments: This command compares the contents of files to determine whether or not they are identical. If filespec1 and filespec2 are individual file names, containing no wild cards, then the contents of the two files are compared. If filespec1 contains wild card symbols and filespec2 does not, then each file specified in filespec1 is compared with the file named in filespec2 that meets the criteria specified below. If both filespec1 and filespec2 contain wild cards, then only files with the same name are compared.

COMP will not compare two files if their sizes are different, unless the /N=num switch is used, since they could not possibly be identical. In this case, DOS displays an error message and asks if you want to compare more files.

If a parameter is missing or does not adequately specify a set of files, DOS will prompt you for the missing information.

If you specify a floppy disk drive in one of the filespecs, DOS assumes the drive is ready. You should therefore be sure the drive has the necessary disk in it before entering this command.

If two files that are being compared do not match, DOS displays an erro message in the following format:

```
Compare error at OFFSET xxx
File1 = yy
File2 = zz
```

Where **xxx** indicates the number of bytes (in hexadecimal) into the files at which a mismatch occurred, **yy** indicates the hexadecimal value of the first mismatched byte in file1, and **zz** indicates the hexadecimal value o the first mismatched byte in file2. If COMP encounters 10 files that do not match it terminates and displays a message.

If the message “EOF mark not found” appears it means that the files may be identical even if they produced “Compare error” messages. This occurs because some types of files have directory entries that indicate they are a multiple of 128 bytes in length, when in fact they terminate before the end of the last 128 byte block. COMP still attempts to compare these files out to the end of the last 128 byte block, and it may be that the Compare error occurred beyond the true end of the file. In that case, the files would be identical but still produce an error message.

Note that this command is distinctly different from the DISKCOMP command, which is used in the Shell DOS Utilities Disk Compare option DISKCOMP compares two complete disks to see if they are identical in all respects, while COMP compares the contents of two files.

COPY

Type: Internal.

Purpose: Used to copy one or more files from one location to another. Also used to combine two or more files into one file.

Syntax:

```
COPY filespec1      /B filespec2      /B    /V
                    /A                /A

COPY f1 + f2 + ... + fn   /A target   /A /V
                          /B          /B
```

Parameters: `filespec1` specifies the file or files to be copied. Wild card symbols may be used.

`filespec2` specifies the destination of the new copies. Wild card symbols may be used.

`/A` and `/B` are optional parameters. They are used to tell DOS where in the source file to stop copying, and how to specify the end of the destination file. When /A is specified for the source file, DOS expects the source file to be an ASCII text file and copies everything up to the first control-Z character it encounters (control-Z is the ASCII end-of-file character). The control-Z character itself is not copied. If neither /A nor /B is specified, /A is taken as the default if two or more files are being combined.

When /B is specified for the source file, DOS determines the size of the file by the number of bytes specified for it in the directory. If neither /A nor /B is specified, /B is taken as the default unless two or more files are being combined (see Comments below).

When /A is specified for the target file, DOS adds a control-Z character to the end of the file.

When /B is specified for the target file, DOS does not add a control-Z character to the end of the file.

`/V` makes absolutely certain that the new file is written to the disk without any errors. The V stands for Verify. This slows down the copy process considerably, since DOS must read and check each sector after it is written. In most cases, it is unnecessary since errors rarely occur during a copy process, but if the occurrence of an error will result in a serious loss then it is worth the extra time.

`f1, f2, fn` represent files that will be combined to create one new file, specified by `target`.

Comments: All file specifications (indicated in the syntax by filespec1, filespec2, f1 f2, fn, and target) may include a drive and path specification. COPY copie only files in the specified directory, or in the current directory if no directory is specified. Files in subdirectories are not copied. If you wan to copy files in subdirectories you should use the BACKUP or XCOPY command.

If the target directory (filespec2 and target in the syntax) is not the same as the source directory the files will be copied with their same names unless a new name is specified. If the target directory is the same as the source directory then new names must be specified, since a file cannot be copied to itself.

If the target file name already exists, the old file is replaced by the new one. DOS does not issue a warning before replacing the old one, so you must be careful not to overwrite a file you want to keep.

The target file will never be created with the read-only attribute set on even if the source file has read-only set on.

You may use COPY to transfer information between devices. For example the command

```
COPY CON: filename
```

will copy whatever you type at the keyboard into the file named file name Copying will end when you press Ctrl-Z, then Enter←. This will create a file named filename that contains whatever you typed at the keyboard up to the control-Z. The command

```
COPY filename PRN:
```

will copy the file `filename` to the printer, causing it to be printed.

When combining files, if you do not specify a target file, then all files following the first one named are appended to the first one named. For example, the command

```
COPY file1 + file2 + file3
```

will result in the files `file2` and `file3` being appended to `file1`.

COUNTRY

Type: CONFIG.SYS

Purpose: Specifies the country that country-specific formats are used for. Formats include date and time formats, collating sequence, capitalization, folding format, currency symbol, and decimal separator.

Syntax:

```
COUNTRY=code, codepage, filespec
```

Parameters: `code` refers to a three-digit number that specifies a particular country, as specified in Table 10-1.

`codepage` refers to the three-digit code page to use with the specific country. Code pages determine how the ASCII codes will be translated into characters. Note that some country codes support more than one code page. The code pages for each country are specified in Table 10-1. The code page is optional in the COUNTRY command. If it is not used DOS will use the primary code page for the specified country. The primary code page is the first one listed in Table 10-1 (the primary code page for the U.S. is 437).

`filespec` indicates the file that contains the country-specific information. In DOS 5.0, this file is named COUNTRY.SYS. If the filespec parameter is omitted, DOS assumes that COUNTRY.SYS is located in the root directory of the boot disk. If DOS cannot locate the file, it produces an error message when it encounters the COUNTRY command.

Comments: If you specify the filespec parameter you must use the code page parameter, or replace it with a comma so that DOS will know not to interpret the filespec as a code page. For example:

```
COUNTRY=001,,C:\DOS\COUNTRY.SYS
```

Notice that two commas are used — one replaces the code page, and the other is the command that follows the code page parameter.

If no COUNTRY command in included in the CONFIG.SYS file, the default country code that DOS uses is 001, which is the U.S. country code, and code page 437.

It is important to understand that the COUNTRY command does not translate DOS messages or other information into another language, it simply defines the format for such information as date and time.

Table 10-1 — Country Codes and Code Pages.

Country	Code	Available Code Pages
Belgium	032	850, 437
Brazil	055	850, 437
Canada (French-speaking)	002	863, 850
Denmark	045	850, 865
English, Int'l.	061	437, 850
Finland	358	850, 437
France	033	437, 850
Germany	049	437, 850
Italy	039	437, 850
Latin America	003	437, 850
Netherlands	031	437, 850
Norway	047	850, 865
Portugal	351	850, 860
Spain	034	437, 850
Sweden	046	437, 850
Switzerland	041	850, 437
United Kingdom	044	437, 850
United States	001	437, 850

DATE

Type: Internal.

Purpose: Sets the date that is stored in the internal clock.

Syntax:

```
DATE mm-dd-yy
DATE dd-mm-yy
DATE yy-mm-dd
DATE
```

Parameters: `mm` stands for the numeric month (1 - 12)

`dd` stands for the day of the month (1 - 31)

`yy` stands for the year. yy may be entered as a two-digit or four-digit year. If entered as a four-digit year, the century must be 19 or 20.

Comments: If you type an incorrect date, DOS will display the message

```
Invalid date Enter new date (mm-dd-yy)
```

If you press Enter← at this point, the date will be left unchanged.

You may use the forward slash (/) in place of the hyphens to separate the month, day and year.

Entering DATE with no parameters causes DOS to display the current date. A three-letter abbreviation of the day of the week will also be displayed:

```
Sun 10-16-1988
```

The Shell DOS Utilities group provides a menu selection for setting the date and time.

DEL

Type: Internal.

Purpose: Deletes one or more files from a disk.

Syntax: `DEL filespec /P`

Parameters: `filespec` is the name of the file(s) that you want to delete. Wild cards may be used. A drive and path specifier may be included.

`/P` causes DOS to display each file name and pause with a prompt that asks whether you want to delete the file before actually deleting it from the disk.

Comments: The DEL command is functionally identical to the ERASE command.

DEL will not remove files whose Hidden attribute is set to ON.

If the DEL command is entered with a drive and path specifier, but no file name, it assumes *.* for the file name.

If you enter a DEL command to delete all file in a directory or all files on a disk, and do not use the /P parameter, DOS will pause and display the message

```
All files in directory will be deleted!
Are you sure (Y/N)?
```

This is to prevent you from accidentally deleting a large number of files by accident. Press either Y for Yes or N for No and press Enter←.

The DEL command cannot be used to delete a subdirectory, you must use the RMDIR (Remove Directory) command for that purpose.

The Shell File System provides much more convenient and powerful features for removing files and subdirectories than the DEL command.

DEVICE

Type: CONFIG.SYS

Purpose: DEVICE is used to install a device driver when you boot your computer.

Syntax: `DEVICE=filespec parameters`

Parameters: `filespec` is the file name of the device to be installed. It may include a drive and path specification.

`parameters` represents any parameters required by the device driver.

Comments: A separate `DEVICE=` command is required for each device driver you want to install.

DEVICEHIGH

Type: CONFIG.SYS

Purpose: DEVICEHIGH is used to install a device driver into a UMB (Upper Memory Block) when you boot your computer.

Syntax: `DEVICEHIGH size=hexsize=filespec parameters`

Parameters: `size=hexsize` is an optional specification of the minimum amount of memory that must be available in a UMB before DEVICEHIGH will attempt to load the driver into high memory.

`filespec` is the file name of the device to be installed. It may include a drive and path specification.

`parameters` represents any parameters required by the device driver.

Comments: For DEVICEHIGH to work, three conditions must be met: you must have at least one meg of memory installed in your PC; you must have an upper memory manager installed; and the command DOS=UMB must appear in the CONFIG.SYS file. If you have a 386 or later CPU, then the EMM386.EXE memory manager that is provided with DOS 5.0 can be used to manage the upper memory area. If you have an earlier CPU, you will have to purchase an upper memory manager that is suitable for your computer.

DEVICEHIGH can save conventional memory resources by loading device drivers into high memory. However, some device drivers will not work in high memory.

This command is discussed in detail in Chapter 4.

DIR

Type: Internal.

Purpose: Displays a directory listing for a specified directory.

Syntax: `DIR filespec /P /W /A:attrib /O:sortorder /S /B /I`

Parameters: `filespec` specifies the directory to be displayed. A drive and path may be included. A file name and wild cards may be used to display only selected files. If no filespec is provided the current drive is assumed, and the file name *.* is assumed.

`/P` stands for pause. It is an optional parameter. When it is used, after one screen is displayed DOS pauses with the message "Press any key to continue ..." and waits for a key to be pressed before displaying the next screen.

`/W` stands for wide. It is an optional parameter. When it is used, it eliminates the size, date, and time information from the display and places the file names in columns across the screen, thereby allowing many more file names to be displayed on one screen. /W should only be used with 80-column monitors.

`/A:attrib` where attrib stands for A, H, R, or S, is used to display only those files with the specified attribute.

`/O:sortorder` can be used to sort the file names in a variety of ways.

`/S` is used to include subdirectories in the search.

`/B` is used to specify a brief listing: one file name per line and no heading or summary information. /B overrides /W.

`/L` displays file names in lowercase, unsorted.

Comments: This command is discussed in detail in Chapter 5.

Files that have the Hidden attribute set to ON are not displayed in DIR listings.

Subdirectories are shown, as well as files. Subdirectories are labeled with <DIR> in the file size area.

If a file name does not have a period, and you want to specify the name in the DIR command, you should terminate it with a period. For example, if a file was named MYFILE the following command would be used to display its directory entry:

```
DIR MFYILE.
```

In addition to the files contained in the specified directory, the DIR command displays the volume label and serial number, if there is one. The number of bytes free on the specified drive is shown at the end of the file listing.

DISKCOMP

Type: External.

Purpose: Compares two floppy disks to determine whether they are identical in all respects.

Syntax: `DISKCOMP source target /1 /8`

Parameters: `source` is the source drive to compare to target.

`target` is the target drive against which source is compared.

`/1` tells DOS to compare only the first side of the disks even if they are double sided.

`/8` tells DOS to compare only 8 sectors per track, even if the source disk has 9 or 15 sectors per track.

Comments: DISKCOMP compares two floppy disks sector by sector. If two disks do not compare exactly, DOS will display a message indicating the track and side at which they first differed.

If two disks contain identical sets of files (as determined by the COMP command), but the files do not occupy identical sectors on each disk, the disks will not compare as identical with the DISKCOMP program. For example, on the source disk FILE1 may occupy sectors 1, 2, and 3 of track 1, while on the target disk FILE1 may occupy sectors 4, 5, and 6 of track 2.

If you have only one floppy drive, the DISKCOMP command can be used by omitting the target drive specification. DISKCOMP will then compare two disks on the same drive, prompting you to change disks at the appropriate times.

Unless you use the /1 or /8 parameters, DISKCOMP will automatically determine the maximum number of sides and sectors per track supported by the source drive and uses those values.

The Shell DOS Utilities menu includes a Disk Compare program that uses DISKCOMP. This is the easiest way to access DISKCOMP.

DISKCOPY

Type: External.

Purpose: Used to create an exact duplicate of a floppy disk.

Syntax: `DISKCOPY source target /1 /V`

Parameters: `source` is the drive to be copied from.

`target` is the drive to be copied to.

`/1` copies only side 1 of the source disk, even if it is a two-sided disk.

`/V` is used to verify the copy. This slows down the operation.

Comments: DISKCOPY creates a copy of a disk that is identical in all respects. A disk that has been created with DISKCOPY will DISKCOMP successfully with the original disk.

If the target disk is not formatted, DISKCOPY will format it during the copy process.

Any existing files on the target disk will be erased during a DISKCOPY.

If you have only one floppy drive, omit the target drive. DOS will then perform the copy using a single drive. You will be prompted when to change disks.

If an error occurs during the copy, DOS displays a warning message indicating the drive, side and track on which the error occurred, then proceeds to complete the copy.

DISKCOPY will ignore any ASSIGN commands that are in effect.

In general, it is better to use the COPY command, since this will unfragment files, and will not erase any additional files on the target disk.

DOS

Type: CONFIG.SYS

Purpose: Used to load DOS into the High Memory Area (HMA), and/or provide access to the Upper Memory Blocks (UMBs) for use by the LOADHIGH and DEVICEHIGH commands.

Syntax: `DOS=HIGH/LOW, UMB/NOUMB`

Parameters: `HIGH/LOW` specifies whether DOS should go into high or low memory.

`UMB/NOUMB` specifies whether DOS should make Upper Memory Blocks available for loading device drivers and TSRs.

Comments: This command is discussed in detail in Chapter 4.

DOSKEY

Type: External.

Purpose: Loads the DOSKEY program into memory.

Syntax: `DOSKEY /REINSTALL /BUFSIZE=size /MACROS /HISTORY /INSERT or OVERSTRIKE /MACRONAME=text`

Parameters: `/REINSTALL` installs a new copy of DOSKEY; the memory used for the previous copy is lost.

`/BUFSIZE=size` sets up the size of the command stack/macro buffer.

`/MACROS` displays all defined macros.

`/INSERT` or `OVERSTRIKE` specifies the typing mode.

`/MACRONAME=text` creates a macro.

Comments: This command is discussed in detail in Chapter 3.

DOSSHELL

Type: External.

Purpose: Starts the DOS 5.00 Shell from the command line.

Syntax: `DOSSHELL /T:res lines /B DOSSHELL /G:res lines /B`

Parameters: `/T:res lines` starts the Shell in Text mode, at the specified resolution, with the specified number of lines on the screen.

`/G:res lines` starts the Shell in Graphics mode, at the specified resolution, with the specified number of lines on the screen.

`/B` starts the Shell in black and white.

Comments: This command is discussed in detail in Chapter 2.

ECHO

Type: Internal, Batch.

Purpose: Turns on and off the screen display of batch file commands. Also used to force the display of a message in a batch file or Shell program.

Syntax:

```
ECHO ON
ECHO OFF
ECHO message
ECHO
```

Parameters: `ON` turns on ECHO mode. ON is the default mode.

`OFF` turns off ECHO mode.

`message` is a message that you want displayed, even if ECHO is set off.

Comments: When the ECHO command is entered with no parameters, the current ECHO state is displayed.

`ECHO ON` and `ECHO OFF` are useful in batch files for causing or suppressing the display of batch file commands. They have no effect in Shell programs since the Shell automatically suppresses all Shell commands.

ECHO message displays the specified message. This command is useful in both batch files and Shell programs.

EXPAND

Type: External.

Purpose: Expands a compressed file from the original DOS 5.0.

Syntax:

```
EXPAND source target
```

Parameters: `source` is the file specifier for the original DOS 5.0 file. These files all have extensions that end with an underscore (_).

`target` is the location that the uncompressed file should be stored. You may specify a new name, or leave the name of the compressed file. The extension will automatically be corrected.

Comments: The original DOS 5.0 disks contain all programs in a compressed format. During the SETUP operation, in which DOS 5.0 is installed, the appropriate files are expanded. However, if you change the configuration of your computer, you may need to add some files that were not loaded during the original setup. A typical example is taking a PC that has a monochrome or CGA video card installed, and later adding an EGA or VGA card. The DOS 5.0 Shell will require an additional driver file, which must be expanded from the original disks.

FASTOPEN

Type: External and CONFIG.SYS

Purpose: Increases the speed with which you can access disk files, when the same files are used repeatedly.

Syntax:

```
FASTOPEN drive1=buffers drive2=buffers .../X
FASTOPEN drive1=(buffers,continuous-buffers)... /X
FASTOPEN drive=(buffers,)... /X
FASTOPEN drive=(,continuous-buffers)... /X
INSTALL=FASTOPEN.EXE ...
```

Parameters: `drive` is the drive on which the files to be made quickly accessible reside.

`buffers` specifies the number of buffers to be used for the specified drive. The minimum number is 10 and the maximum is 999 for all drives specified. If buffers is not specified with FASTOPEN the default value is 34.

`continuous-buffers` is the number of continuous space buffers to be used for the specified drive. Continuous-space buffers record information related to the way a file's contents are distributed on the disk. The minimum number is 1 and the maximum number is 999 for all drives specified. If only a drive letter is specified with FASTOPEN the default value is 34. If buffers is specified but continuous-buffers is not, the continuous-buffers feature is not activated.

`/X` causes DOS to use expanded memory for the FASTOPEN buffers. This saves conventional memory space for use by application programs.

Comments: FASTOPEN works by setting up buffers in memory in that contain the directory information for recently opened files. This information includes the location of the directory information and continuous space information. Retaining this information in a memory buffer allows it to be retrieved much faster than if it is read from the disk.

There is one drawback to using FASTOPEN: it reduces the amount of memory available to applications, unless you have expanded memory and specify the /X parameter. Each buffer takes 35 bytes of memory, and each continuous-buffer takes 16 bytes of memory.

The `INSTALL=` version of the command is used if you want to execute FASTOPEN from the CONFIG.SYS file during boot-up. You must include the extension (.EXE) if this version is used.

All drives for which you want to specify FASTOPEN buffers must be specified the first time you use FASTOPEN.

The number of buffers specified should be larger than the greatest number of subdirectories that share a common path. For example, if the path to your deepest subdirectory contained 10 directories, then there should be at least 10 buffers.

Example: `FASTOPEN C:=(100,200)` specifies 100 buffers for file and directory information, and 200 buffers for continuous space information.

FC

Type: External.

Purpose: Finds and displays the differences between two files.

Syntax:
```
FC /A /C /L /LBn /N /T /W /nnn drive1:filespec1
drive2:filespec2
```

Parameters: `/A` abbreviates the output when comparing in ASCII mode. Without this switch, if 10 consecutive lines are different, then FC displays all 10 lines. With /A, only the matching lines that surround the consecutive mismatching lines are displayed.

`/C` turns off case sensitivity.

`/L` forces the comparison into ASCII mode. For files that have any extension other than BIN, COM, EXE, LIB, OBJ, and SYS, ASCII mode is the default and /L has no effect. For files that have one of these extensions, binary mode is the default and the /L switch is needed for an ASCII comparison.

`/LBn` adjusts the internal line buffer to **n** lines.

`/N` displays line numbers when performing an ASCII comparison.

`/T` turns off tab expansions. Normally, tabs are expanded to eight spaces.

`/W` converts tabs and multiple blanks (white space) to a single blank. Blanks at the beginning of a line are ignored.

`/nnn` specifies how many consecutive lines must match for FC to consider the files resynchronized.

`/B` forces the comparison into binary mode. This is the default mode for files that have an extension of BIN, COM, EXE, LIB, OBJ, and SYS. All other files require this switch to be compared in binary mode.

Comments: FC is used to identify the differences between two files, where COMP is used to verify that two files are identical. The FC command is discussed in detail in Chapter 5.

FILES

Type: CONFIG.SYS

Purpose: Sets the maximum number of files that can be open at any one time.

Syntax: `FILES=num`

Parameters: `num` specifies the maximum number of files that can be open at any one time. The minimum is 8 and the maximum is 255.

Comments: If no `FILES=` command is included in your CONFIG.SYS file the default value of 8 is used.

Each time a program is run the file that contains the program is opened. Each data file that must be accessed by a program it must be opened. If a program uses overlay files, they will also have to be opened. It is becoming common for applications to require more than eight files to be open at once. FILES is used when more than eight files must be open at one time.

FIND

Type: External.

Purpose: Locates text strings in files.

Syntax: `FIND /C /N /V /I "string" filespec1 filespec2 ...`

Parameters: `/C` displays a count of the lines that were found to contain the string.

`/N` displays the line number and the line that was found to contain the string.

`/V` displays all lines that do not contain the string.

`/I` makes the search insensitive to case.

`filespec1`, `filespec2`, etc. specify the files to be searched for the string. They cannot contain wild cards, but drive and path specifiers are allowed.

Comments: Files are assumed to be ASCII text files, so they are searched up to the first control-Z character encountered.

The string to be searched for must be enclosed in double quotes.

FOR

Type: Internal.

Purpose: Used in batch files and at the command line for repetitively executing a command.

Syntax: `FOR %variable IN (set) DO command FOR %%variable IN (set) DO command`

Parameters: `%variable` is the % sign followed by any single character, including 0 through 9. `%variable` is used in batch files to indicate a variable name that will take on each value in `(set)`, one at a time, and execute command for each value.

`%%variable` is the same as `%variable`, except it is used at the command line instead of in a batch file.

`(set)` contains one or more values that the variable takes on, each for one execution of the command.

`command` represents the DOS command that will be executed once for each value in `(set)`.

Comments: FOR is useful when a single command must be executed several times, with one value in the command changing. The following command entered at the command prompt will display the directory listing of DIR1, DIR2 and DIR3:

```
FOR %D IN (DIR1, DIR2, DIR3) DO DIR %D
```

The same command could be executed at the command prompt as follows:

```
FOR %%D IN (DIR1, DIR2, DIR3) DO DIR %%D
```

FORMAT

Type: External.

Purpose: Prepares a new floppy or fixed disk for use.

Syntax:

```
FORMAT drive: /Q /U /S /B /V:label /8 /4 /1 /N:xx /T:yy /F:size
```

Parameters: `drive` is the drive on which the disk will be formatted.

`/Q` performs a quick format — the FAT and root directory are replaced, but the disk is not scanned for bad sectors.

`/U` specifies an unconditional format. This does not create a mirror image of the FAT and root directory before formatting. Therefore, it makes the disk difficult or impossible to unformat.

`/S` copies the operating system files to the disk so that the computer may be booted from the disk

`/B` formats a floppy disk and reserves space for the system files, so that even if files are copied to the disk, the system files may still be transferred to it using the SYS command.

`/V:label` specifies a volume label for the disk. If /V is not specified, you will be prompted to enter a label after the format is complete. If you press Enter↵, no label will be created.

`/8` is used to format a 5.25-inch floppy disk with eight sectors per track (as opposed to the default of 9).

`/4` is used to format a 360KB or 180KB 5.25-inch floppy disk in a high capacity (1.2 Meg) drive. This does not produce a disk that can be reliably read or written in a 360KB drive.

`/1` format side one only on a 5.25-inch floppy disk.

`/N:xx /T:yy` specifies the number of sectors per track and the number of tracks per disk. If either parameter is used, then both parameters must be specified or DOS will terminate the command with an error message.

`/F:size` allows you to specify an alternate media size for a floppy disk. Valid values for `size` are: 160K, 180K, 320K, 360K, 720K, 1200K, 1440K, and 2880K.

Comments: In DOS 5.0, the normal format is a safe format, which means that a mirror image of the FAT and root directory are copied to the disk before the format is carried out. This allows the UNFORMAT command to recover a disk that has been accidentally formatted.

The /Q switch should only be used on disks that have previously been formatted, and which you are sure contain no bad clusters.

The /U switch turns the safe format off, and is therefore a little faster, since no mirror image is created.

GOTO

Type: Internal.

Purpose: Used in batch files to create a branching flow of control.

Syntax: `GOTO label`

Parameters: `label` refers to a line in the file that contains a colon followed by the label. When the GOTO command executes, control jumps to the line following the label.

Comments: Only the first eight characters of a label are examined, so if more than one label is used in a batch file, they must not contain the same first eight characters.

Labels are never displayed, even if ECHO is set to on, so you may use the colon to add comments to a batch file instead of the REM command, which is echoed unless ECHO is set to off. If you use the colon in this way make sure the first eight characters of the comments do not match the first eight characters of a legitimate label.

GRAPHICS

Type: External.

Purpose: Used to print graphics images.

Syntax:

```
GRAPHICS printer-type drive:path\profile /R /B /LCD
/PB:identifier
```

Parameters: `printer-type` indicates the type of printer you have. If you do not specify a printer type, the default value is GRAPHICS.

`profile` refers to the file that contains information on all printers that are supported by the GRAPHICS command. If the profile parameter is omitted, the default value is GRAPHICS.PRO.

`/R` specifies that black and white should not be reversed. Normally everything that appears black on the screen is not printed, and everything that appears white is printed in black. /R is used to turn off this reversal.

`/B` causes the background color to print, if you have a color printer specified. If /B is not included in the GRAPHICS command then the background color is not printed.

`/LCD` is used if you have a liquid crystal display and want to print the graphics image exactly as it appears on the display. This parameter has the same effect as specifying `/PB:LCD`.

`/PB:identifier` is used if you have a laptop computer. It specifies how to print screen images when different displays are used on the computer. `Identifier` can be either `STD`, which specifies a standard full-size screen, or `LCD`, which specifies a liquid crystal display. The aspect, or ratio of width to height, is different on an LCD display.

Comments: Your printer must have the ability to print graphics images for this command to work.

HELP

Type: External.

Purpose: Provides online help for DOS commands.

Syntax: `HELP command`

Parameters: command is any DOS command.

Comments: `HELP` entered by itself provides a brief description of each DOS command. When a command is passed as a parameter, more extensive help is provided.

An alternative way to get help for a specific command is to append the /? switch to the command. For example, `DIR /?` will display a help screen for the DIR command. This is a little faster than the equivalent HELP DIR form.

IF

Type: Batch.

Purpose: Provides logical IF – THEN programming logic in batch files.

Syntax: `IF (NOT) condition command`

Parameters: `NOT` is used to cause the command to execute only if the condition is not met.

`condition` may be one of the following:

1. The existence of a file that is named in the batch file.
2. The existence of a file passed as a parameter.
3. A parameter matches a value that is specified in the batch file.
4. Whether the previous command in the batch file terminated normally.

`command` is any DOS command.

Comments: The IF command is discussed in detail in Chapter 8.

INSTALL

Type: CONFIG.SYS

Purpose: Can be used in the CONFIG.SYS file to load TSR programs without storing a copy of the environment with the program.

Syntax: `INSTALL=command`

Parameters: `command` is a DOS or other program command for a TSR program.

Comments: Many TSRs require a copy of the DOS environment, and therefore cannot be loaded with the INSTALL command. The MS-DOS TSRs that can be used with INSTALL include FASTOPEN, KEYB, NLSFUNC, and SHARE.

JOIN

Type: External.

Purpose: Makes one drive appear to be a subdirectory of another drive.

Syntax: `JOIN drive1 drive2:path JOIN drive/d`

Parameters: `drive1` is the drive that will appear as a directory on `drive2`.

`drive2:path` is the drive and optional path that will refer to `drive1`.

`drive/d` deletes the join for the specified drive. The `drive` referenced here should have been `drive1` in an earlier JOIN command.

Comments: When used with no parameters, JOIN displays a list of joins in effect.

The directory specified for `drive2` must either not exists or be empty. If it does not exist, an empty directory of the specified name is created, and left on `drive2` after the JOIN is ended. If the directory does exist, and is not empty, JOIN will display the message "Directory not empty" and terminate without forming the join.

When a join is in effect, references to `drive1` will not be recognized, and will generate the message "Invalid drive specification."

The following commands do not work with a drive formed by JOIN: BACKUP, CHKDSK, DISKCOMP, DISKCOPY, FDISK, FORMAT, LABEL, MIRROR, RECOVER, RESTORE, and SYS.

KEYB

Type: External and CONFIG.SYS

Purpose: Defines a keyboard layout that corresponds to that used in a country other than the U.S.

Syntax:

```
KEYB layout codepage filename /E /ID:identifier

KEYB identifier

KEYB

INSTALL=KEYB.COM ...
```

Parameters: `layout` specifies the layout as defined in the following table.

`codepage` specifies the code page to use. Two code pages are available for each country. If `codepage` is not specified the first code page listed in the following table is used.

`filename` is the name of the file that contains the keyboard definitions. If filename is omitted KEYB looks in the root directory of the current drive for the file named KEYBOARD.SYS.

`/E` indicates that an enhanced keyboard is installed. When the computer is an 8088/8086, DOS assumes it is not using an enhanced keyboard, so if you are using an enhanced keyboard with one of these PCs, /E is required. For later model computers, an enhanced keyboard is assumed and, therefore, the /E switch is not required.

`ID:identifier` specifies an alternate keyboard for the country specified. In the table below, some countries have two keyboards available, and the first listed is used by default. Specifying the second with the `ID:identifier` will cause that keyboard to be used instead.

The following table specifies the values that the various parameters may take for each country.

Country	Keyboard Layout	Code Pages	Identifier
Belgium	BE	850, 437	
Brazil	BR	850, 437	
Canada (French speaking)	CF	850, 863	
Denmark	DK	850, 865	
Finland	SU	850, 437	
France	FR	850, 437	120, 189
Germany	GR	850, 437	
Italy	IT	850, 437	141, 142
Latin America	LA	850, 437	
Netherlands	NL	850, 437	
Norway	NO	850, 865	
Portugal	PO	850, 860	
Spain	SP	850, 437	
Sweden	SV	850, 437	
Switzerland (French speaking)	SF	850, 437	
Switzerland (German speaking)	SG	850, 437	
United Kingdom	UK	850, 437	168, 166
United States	US	850, 437	

Comments: The INSTALL version of the command is used when the KEYB command is included in the CONFIG.SYS file.

When `KEYB` is entered without any parameters the current alternate keyboard is displayed.

You may change the alternate keyboard by entering the `KEYB` command. When the alternate keyboard is in effect, pressing Ctrl-F1 will switch to the U.S. keyboard. Pressing Ctrl-F2 will switch back to the alternate keyboard.

Example: `KEYB FR,437 /ID:189` specifies the French keyboard layout, using code page 437 and keyboard ID 189

LABEL

Type: External.

Purpose: Defines a disk volume label. If a disk does not have a label, the LABEL command adds one. If the disk does have a label, the LABEL command is used to change or delete the label.

Syntax: `LABEL drive label`

Parameters: `drive` is the drive containing the disk whose label will be defined. If the drive is not specified, the current drive is assumed.

`label` is the label that will be placed on the disk. The label can be from 1 to 11 characters. The same characters as used for file names can be used, but no period should be used to separate an extension. Spaces may also be used.

Comments: If no label is specified a message similar to one of the following messages will appear:

```
Volume label in drive C is label
Volume Serial Number is 5A32-21E3
Volume label (11 characters, ENTER for none)?
```

```
Volume label in drive C has no label
Volume label (11 characters, ENTER for none)?
```

LASTDRIVE

Type: CONFIG.SYS

Purpose: Defines the maximum number of drives that may be accessed.

Syntax: `LASTDRIVE=driveletter`

arameters: `driveletter` is any alphabetic character from A to Z. Upper and lowercase are treated identically. This is the last drive letter that DOS will allow you to reference.

omments: If no LASTDRIVE command is included in your CONFIG.SYS file the default letter E is used.

LASTDRIVE is useful when used with the SUBST command. It determines the last drive letter you may use when SUBST is in effect.

The minimum drive letter you may specify is the last drive letter on your computer. If you specify a drive letter less than the last drive letter on your computer, the command will be ignored when CONFIG.SYS commands are executed.

OADHIGH or LH

ype: Internal.

urpose: Loads a program into high memory. This conserves conventional memory for use by other programs.

yntax:

```
LOADHIGH filespec

LH filespec
```

arameters: `filespec` is the file specifier of the program to load.

omments: For LOADHIGH to work, three conditions must be met: you must have at least one meg of memory installed in your PC; you must have an upper memory manager installed; and the command `DOS=UMB` must appear in the CONFIG.SYS file. If you have a 386 or later CPU, then the EMM386.EXE memory manager that is provided with DOS 5.0 can be used to manage the upper memory area. If you have an earlier CPU, you will have to purchase an upper memory manager that is suitable for your computer.

This command is discussed in detail in Chapter 4.

MEM

Type: External.

Purpose: Displays a report of the amount of memory used by the system, the contents of memory in use, and the amount of memory available for programs.

Syntax: `MEM /PROGRAM /DEBUG /CLASSIFY`

Parameters: `/PROGRAM` is used to list all programs currently in memory and the total amounts of memory in the system, in use, and available.

`/DEBUG` is used to list much more detailed information regarding the current contents of the system memory. This includes system device drivers, installed device drivers, all programs currently in memory, and the total amounts of memory in the computer, in use, and available.

`/CLASSIFY` lists all programs in memory, and shows how much memory they occupy. This switch also provides a summary of how memory is being used, and lists the both the total available memory and the largest single block of memory available. If high memory has been made available, lists what is stored there, how much is available, and what the largest UMB is.

Comments: Entering the MEM command without any parameters produces just a report of used and unused memory, similar to the following:

655360 bytes total memory 655360 bytes available 569504 largest executable program size

1703936 bytes total EMS memory 1703936 bytes free EMS memory 524288 bytes total extended memory 524288 bytes available extended memory

The following is an example of the result of entering the MEM command with the `/PROGRAM` parameter:

```
Address  Name      Size    Type

000000             000400  Interrupt Vector
000400             000100  ROM Communication Area
000500             000200  DOS Communication Area
000700   IBMBIO    002130  System Program
002830   IBMDOS    008890  System Program
00B0C0   IBMBIO    006A00  System Data
         ANSI      001190  DEVICE=
                   0000C0  FILES=
                   0004B0  FCBS=
                   004410  BUFFERS=
                   0001C0  LASTDRIVE=
                   000CD0  STACKS=
011AD0   IBMDOS    000030  -- Free --
011B10   KBDRIV    000040  Environment
011B60   IBMDOS    000030  -- Free --
011B60   IBMDOS    000030  -- Free --
011BA0   FASTOPEN  0041B0  Program

   655360 bytes total memory
   655360 bytes available
   302656 largest executable program size

TIME = 18:43:14.23 DATE = Sat 06-01-1991
```

The MEM command with the /DEBUG parameter produces a report similar to that produced with /PROGRAM, only with a finer breakdown of the use of memory.

These switches are all mutually exclusive.

MIRROR

Type: External.

Purpose: The MIRROR program is used to create Mirror files, which are snapshots of the FAT and root directory and optionally the partition table. These files can be used to rebuild a disk after an accidental formatting, or after the RECOVER command has been used to recover a damaged disk. MIRROR may also be used as a TSR to create a deletion tracking file for undeleting files.

Syntax:

```
MIRROR DRIVE: /1 /TDRIVE-ENTRIES
```

```
MIRROR /U
```

```
MIRROR /PARTN
```

Parameters: `DRIVE` is the drive that has the disk for which mirror files should be created.

`/1` tells MIRROR to overwrite the previous mirror file, if one exists. Without this switch, the previous file is renamed to MIRROR.BAK.

`/TDRIVE-ENTRIES` loads the TSR part of MIRROR that creates the deletion tracking file. DRIVE is the drive for which deletion tracking should be installed, entered without a colon. ENTRIES is an optional number that indicates the number of deletions to track.

`/U` unloads the deletion-tracking TSR, thereby disabling deletion tracking.

`/PARTN` is used to create a mirror file of the partition table.

Comments: The UNDELETE and UNFORMAT commands use MIRROR to help them restore data. The MIRROR program is discussed extensively in Chapters 5 and 6.

MKDIR or MD

Type: Internal.

Purpose: Creates a new subdirectory on the specified drive, under the specified directory.

Syntax:

```
MKDIR drive:path
MD drive:path
```

Parameters: `drive` indicates the drive on which the new subdirectory should be created. If omitted the current drive is used.

`path` is the complete path specification for the new subdirectory, including the new subdirectory name. If only the new subdirectory name is specified the current directory is used.

Comments: The commands MKDIR and MD are functionally identical.

Naming subdirectories follows the same rules as naming files. It is conventional, however, to omit extensions from subdirectory names.

The number of subdirectories that may exist on a disk is limited only by disk space. Each subdirectory created requires 2048 bytes.

The complete path specification from the root directory to the subdirectory, including the backslash characters required to separate subdirectories, must not exceed 63 characters.

The Shell File System contains a function for creating new directories. It is easier to see exactly what you are doing in the Shell, since you have a graphical representation of the disk directory structure.

MORE

Type: External.

Purpose: Causes the screen display to pause whenever it fills up and waits for a key to be pressed before scrolling up the next screen of information.

Syntax:

```
MORE < filename
command | MORE
```

Parameters: `< filename` causes the contents of the file `filename` to be displayed on the screen in the same way as the TYPE command, except that when the screen fills up, it pauses with the message `-- More--` at the bottom of the screen. Pressing any key will cause the next screen to be displayed.

`command | MORE` causes the output of the command to be processed by MORE, resulting in the display pausing when the screen fills up and the message `-- More--` appearing at the bottom of the screen. Pressing any key will cause the next screen to be displayed.

Comments: An example of the first syntax is:

```
MORE MYFILE.LET
```

This will cause the contents of MYFILE.LET to appear on the screen. MYFILE.LET should be an ASCII text file.

An example of the second syntax is:

```
DIR | MORE
```

This will cause the screen to pause scrolling each time it fills up with directory information. The effect is similar to the /P parameter on the DIR command.

PATH

Type: Internal.

Purpose: Defines a set of directories that DOS will search for executable files if they are not found in the current directory. Executable files are those that have extensions of COM, EXE and BAT. Also used to display the current path setting.

Syntax: `PATH path1; path2; ... PATH ;`

Parameters: `path1`, `path2`, etc. are the directories to be searched. They should include the complete path starting from the root directory.

`;` removes all subdirectories from the path. DOS will only search the current directory for executable files.

Comments: PATH entered with no parameters will display the current path without changing it.

Each time a new path is specified the previous path is removed, so to define multiple subdirectories as part of the path you must include all of them in one PATH command.

If there is an error in a subdirectory specification, DOS will not detect the error until it attempts to access the directory.

If a subdirectory specified in the PATH command does not exist, DOS will ignore that directory and go on to the next.

PAUSE

Type: Internal.

Purpose: The PAUSE command is used in batch files and Shell programs to suspend execution of the batch file or Shell program until a key is pressed.

Syntax: `PAUSE message`

Parameters: `message` is an optional message that will display when the program pauses. This will only display if ECHO is on, and the PAUSE command itself is also displayed.

Comments: PAUSE is useful in batch files and Shell programs, especially in conjunction when the ECHO command is used to display a message instructing the user to perform some action. PAUSE allows them to read the message before the program continues executing.

When the PAUSE command executes it suspends the execution of the batch file or Shell program and displays the message, `Press any key to continue...`

PRINT

Type: External.

urpose: Used to establish and manipulate a print queue for printing one or more files while you continue to use the computer for other purposes.

yntax:

```
PRINT /D:device /Q:num /B:size /S:timeslice /U:busy
/M:max filespec1 /C /T /P filespec2 /C /T /P ...
```

arameters: The parameters are described in detail in Chapter 9.

omments: The PRINT command is discussed in detail in Chapter 9.

PROMPT

Type: Internal.

Purpose: Defines the format of the DOS prompt.

Syntax: `PROMPT text PROMPT`

Parameters: `text` can be constructed of regular text and parameters. The text and parameters may be intermixed as much as desired, though the entire command cannot exceed the DOS limit of 128 characters. The parameters that may be interspersed with text are:

`$$`	Includes a $ in the prompt.
`$t`	Includes the current time in the prompt.
`$d`	Includes the current date in the prompt.
`$p`	Includes the current directory of the current drive in the prompt.
`$v`	Includes the DOS version number in the prompt.
`$n`	Includes the default drive letter in the prompt.
`$g`	Includes a greater than sign (>) in the prompt.
`$l`	Includes the less than sign (<) in the prompt.
`$b`	Includes the bar (|) symbol in the prompt.
`$q`	Includes the equal sign (=) in the prompt.
`$h`	Inserts a backspace into the prompt, deleting the characte immediately preceding the backspace.
`$e`	Includes the Escape character in the prompt. This can be used to send commands to the ANSI.SYS device driver.
`$_`	Includes a Carriage Return/Line Feed sequence in the prompt, allowing you to develop multi-line prompts.

Comments: When PROMPT is entered with no parameters the DOS prompt is rese to its default, which shows the current drive followed by a greater-tha sign.

The prompt is what appears on the DOS command line when you exit the Shell either via a permanent exit (Exit from the Main Group Menu Ba or pressing F3 to exit the Shell) or a temporary exit (Command Prompt from the Main Group or pressing ⇧Shift-F9 to go to the Command Prompt).

The PROMPT command is discussed in detail in Chapter 9.

RECOVER

Type: External.

Purpose: Helps in recovering files that contain occupy a bad disk sector. Also useful in recovering all the files on a disk that has a damaged directory.

Syntax:

```
RECOVER filespec

RECOVER drive
```

Parameters: `filespec` is a file identifier that indicates which file to attempt to recover. It may include a drive and path specification. If no drive or path is specified, the default drive and path are assumed.

`drive` is a drive which contains a disk with a damaged directory.

Comments: If a file contains a bad disk sector, using RECOVER on the file will recover all of the data in the file except that which occurs in the bad sector. For this reason non-text files (program files) are not good candidates for using RECOVER, since you have no reasonable way to restore the missing part of the file.

The same is true of spreadsheet files and database files, since the missing data will probably make at least the last part of the file unreadable by the database or spreadsheet program. However, you may be able to see what the data is and re-enter it.

When a file is recovered, DOS often includes some extra information at the end of the file. This is because DOS recovers complete allocation units, even if the end of the file does not occupy a complete allocation unit.

You are not allowed to recover more files than will fit in the root directory of the disk in one operation. On double-sided floppy disks, this is 112 files. If the total number of files being recovered exceeds this, you must enter multiple RECOVER commands. After the first RECOVER operation, copy the files you want to keep to a new disk and delete them from the damaged disk. Then issue the next RECOVER command, and so on, until all damaged files have been recovered.

The form `RECOVER drive` should only be used if the disk directory has become unusable. DOS will then ignore all information in the directory and attempt to recover everything on the disk. This is done by determining what allocation units belong together to make a file and copying those allocation units into files named FILExxxxx.REC, where **xxxxx** is a sequential number starting with 00001. Each REC file represents one recovered file.

RECOVER is not a miracle program, and it will not necessarily produce useful files. It should be used only in extreme situations, where there is no other choice for recovering files.

REM

Type: Batch files and CONFIG.SYS

Purpose: Used to include lines in batch files and the CONFIG.SYS file that DOS does not attempt to execute, but which contain information helpful to someone reading the file.

Syntax:

```
REM comment
REM
```

Parameters: `comment` is any text message up to 123 characters long that explains some aspect of the commands in the file.

Comments: Using REM by itself helps to space out the lines in a batch file or CONFIG.SYS file, making it easier to read.

If ECHO is on the entire REM line, including the letters REM, will be displayed when DOS encounters the line in a batch file. If ECHO is off, the REM lines will not display on the screen.

RENAME or REN

Type: Internal.

Purpose: Changes the name(s) of one or more files.

Syntax: `RENAME oldname newname`

`REN oldname newname`

Parameters: `oldname` is the name of the file being renamed.

`newname` is the new name of the file being renamed.

Comments: If a file named `newname` already exists, DOS will display a message and terminate the command without changing the file name.

There are two ways in which wild card symbols may be used. The first is to change only the name or only the extension of a file. For example, the command `REN TEST.BAT *.TMP` will rename the file TEST.BAT to TEST.TMP.

The second way in which wild cards may be used is to rename a set of files by replacing only certain letters of their name. For example, the files TEST1.BAT, TEST2.BAT and TEMPEST.BAT could be renamed with the command `REN T*.BAT XY*.BAT` The result of this command is that the three files are given the new names XYST1.BAT, XYST2.BAT, and XYMPEST.BAT. In other words, the number of letters replaced is defined by the number of letters in the new file name.

REPLACE

Type: External.

Purpose: Used to copy files from one disk to another, choosing files based on one or more of several selection criteria as discussed below.

Syntax: `REPLACE source target /P /R /W /S /U /A`

Parameters: `source` is the file specifier of the files to be copied. It may include the * wild card. A drive and path specifier may also be included.

`target` is the target drive to copy to. Target file names are not allowed, since the files will be given the same name as they have on the source.

`/P` causes DOS to prompt you before copying any files that match the specified criteria. This allows you to perform a file-by-file replacement of old files or addition of new files.

`/R` causes DOS to replace files on the target that have the Read-only attribute set ON. If /R is not specified read-only files will not be replaced.

`/W` causes DOS to wait for you to insert a floppy disk into the source drive before it starts to search for the files.

`/S` causes DOS to search subdirectories of the target disk, as well as the current directory, to locate files that match the source file name.

`/U` causes DOS to replace files on the target if the date and time on the source or more current.

`/A` causes DOS to copy all specified files on the source that do not exist on the target. This is used to add files to the target without overwriting existing files. /A may not be used if /S or /U is used.

Comments: If /W is not specified the REPLACE operation will begin as soon as you press Enter↵.

Hidden and system files will not be found on the source and therefore will not be copied to the target.

The REPLACE command returns an error code when it is finished executing. The most common codes are as follows:

Return Code	Cause	Description
2	File not found	No source files exist
3	Path not found	The source path does not exist
5	Access denied	/R should be specified
8	Insufficient memory	REPLACE requires more memory than is available
11	Invalid format	REPLACE command was not entered correctly
15	Invalid drive	A specified drive does not exist
22	Unknown command	The version of DOS being used does not support REPLACE

RESTORE

Type: External.

Purpose: Restores files that have been backed up with the BACKUP command.

Syntax: `RESTORE source target filename /P /M /S /B:mm-dd-yy /A:mm-dd-yy /E:hh:mm.s /L:hh:mm:.s /N /D`

Parameters: `source` is the drive that contains the disk that was backed up with the BACKUP command.

`target` is the drive to which the backed up file(s) should be restored.

`filename` is used to restore only certain files. A drive and path specifier, as well as wild cards, may be included. If no file name is used, the default *.* is used.

`/P` causes the computer to pause before restoring a file that has been modified since the last backup, or has the read-only attribute set to on. This protects against replacing a recent version of a file with an older version.

`/M` is used to restore files that have been modified or deleted since they were last backed up. This allows you to recover an earlier version of a file if the current version has been modified in a way that you don't like.

`/S` restores files in the subdirectories as well as in the current directory.

`/B:mm-dd-yy` restores all files modified on or before the date specified by `mm-dd-yy`.

`/A:mm-dd-yy` restores all files modified on or after the date specified by `mm-dd-yy`.

`/E:hh:mm.s` restores all files modified at or after the time specified by `hh:mm.s`. If /E is used with /B, then files are restored only if they were modified at or before the specified time on or before the specified date.

`/L:hh:mm.s` restores all files modified at or later than the time specified by `hh:mm.s` If /L is used with /A, then only files modified at or later than the time specified, on or after the date specified, are restored.

`/N` is used to restore only files that do not exist on the target. This could be because you are restoring to a disk other than the one that was backed up or because the file has been deleted from the disk.

`/D` displays a list of the files that are stored on the backup disk. No files are restored when /D is specified, but you must still specify a target drive.

Comments: RESTORE only works with files created with the BACKUP command. The system files cannot be restored so RESTORE cannot be used to create a system disk. If you want to create a boot disk, you should use the SYS command to transfer the system files, or format the disk with the /S parameter, before using RESTORE. RESTORE will then transfer the rest of the files and you will have a boot disk that contains the files you want.

The file name may contain drive and path specifiers. If a path is specified a file name must also be specified.

You must RESTORE files to the same directory that they were backed up from. Attempting to restore to a different directory will result in an error message.

When you start RESTORE, you will be prompted to put the disk containing the file(s) to be restored in the source drive. If you are restoring all of the files that were backed up, this should be disk 1. RESTORE will prompt you when to insert subsequent disks. If you are restoring only one file, or a set of files, and know which disk they start with, you may immediately put that disk in the source drive. If you are not sure which disk they start with, put disk 1 in the drive. RESTORE will let you know if the file(s) requested are not there and you can then put in disk 2, and so on.

The RESTORE command sets an Exit code that can be accessed by the Batch command, IF ERRORLEVEL ... The Exit code is set according to the following table:

Exit code	**Meaning**
0	The RESTORE completed normally.
1	There were no files found to restore.
2	Some of the specified files were not restored due to file-sharing conflicts on a network.
3	The RESTORE operation was terminated by the user pressing Ctrl-break.
4	The RESTORE operation terminated due to an error.

RMDIR or RD

Type: Internal.

Purpose: Removes directories from a disk. Only empty directories that contain no files can be removed.

Syntax:

```
RMDIR drive:path
RD drive:path
```

Parameters: `drive` is the drive that contains the directory to be removed.

`path` is the path from the current directory or from the root directory to the directory being removed.

Comments: The root directory cannot be removed. A directory cannot be removed if it is the current directory.

SET

Type: Internal.

Purpose: Creates variables that contain text strings. These variables are stored in the DOS environment and are therefore available to all programs, including batch files.

Syntax:

```
SET variable=value
SET variable=SET
```

Parameters: `variable` is the variable being created.

`value` is the string that is assigned to the variable.

Comments: Variables defined with the SET command can be accessed by batch files in the same manner as %x parameters. For example, the command `SET PW=XYZ` will cause the characters `XYZ` to be substituted for `%PW%` in a batch file.

There is only a limited amount of memory reserved for the DOS environment. If there is not enough environment memory available for the variables you are defining DOS will display an error message.

Entering `SET variable=` with no value will remove the variable from memory. This will free up memory for other variables.

Entering SET with no parameters will display all of the current environment variables.

SETVER

Type: External.

Purpose: Instructs DOS to report a different DOS version number to a program. Can also be used to display and modify the version table, which contains the names of programs and the DOS version number they are designed for. SETVER can add programs to this table.

Syntax:

```
SETVER

SETVER filespec n.nn

SETVER filespec /DELETE /QUIET
```

Parameters: `filespec` identifies the program file to be added to the version table. The file must have an extension of COM or EXE; wild cards are not allowed.

`n.nn` specifies the version number that should be reported to the specified program.

`/DELETE` deletes the version table entry for the specified program. This switch may be shortened to /D.

`/QUIET` causes SETVER to not display any messages when it deletes an entry from the version table.

Comments: When no parameters are used, SETVER displays the contents of the version table. The names of programs contained in the table and the version of DOS that will be reported to them is shown.

Most programs will run unchanged under DOS 5.0. However, some programs have built-in instructions to look for a versions of DOS, and to only run if one of those versions is found. Some of these programs will work perfectly well with DOS 5.0, but they have to be tricked into believing they are running under one of the versions they have listed. They identify the version by sending a query to DOS, so the SETVER command can instruct DOS to report whatever version you like.

If the SETVER command completes satisfactorily, it will display a message warning that you should confirm with the software vendor the compatibility of running the program under DOS 5.0.

You must reboot your system for changes to the version table to take effect.

If you specify a program that is already in the version table, the SETVER command will replace the old one.

SHELL

Type: CONFIG.SYS

Purpose: Can be used to increase the DOS environment size, or to load a command interpreter other than COMMAND.COM.

Syntax:

```
SHELL=filespec parameters
```

Parameters: `filespec` is the name and location of the command interpreter to use.

`parameters` is any parameters and/or switches used by the specified command interpreter.

Comments: The most common use of the SHELL command is to increase the size of the DOS environment. The following command accomplishes this, setting the environment to 512 bytes (the default is 256):

```
SHELL=C:\DOS\COMMAND.COM /E:512 /P
```

SHIFT

Type: Internal.

Purpose: Used in batch files to access more than 10 parameters.

Syntax:

```
SHIFT
```

Parameters: None.

Comments: Each time a SHIFT command is encountered the number of each parameter in a batch file is decreased by 1, the first parameter (%0) is dropped, and an additional parameter can then be used.

The following batch file demonstrates the use of the SHIFT command:

```
DIR%0
PAUSE
DIR%1
PAUSE
DIR%2
PAUSE
DIR%3
PAUSE
DIR%4
PAUSE
DIR%5
PAUSE
DIR%6
PAUSE
DIR%7
PAUSE
DIR%8
PAUSE
DIR%9
PAUSE
SHIFT
DIR%0
PAUSE
DIR%9
```

The first DIR command, `DIR %`, displays the directory listing for the batch file itself. The next nine DIR commands display the directory listings for files specified in the first nine parameters passed on the command line. The `SHIFT` command then shifts all parameter references over one, so that the `DIR %0` command on the third line from the end displays the directory listing for the first parameter, and the `DIR %9` command on the last line displays the first directory listing for the tenth parameter.

The following command demonstrates the use of the batch file, where TEST is the name of the batch file being executed and parm1, parm2, etc. are the parameters being passed to the file:

```
TEST parm1 parm2 parm3 parm4 parm5 parm6 parm7 parm8
parm9 parm10
```

Within the file TEST, %0 refers to the batch file name (TEST), %1 refers to parm1, and so on. The largest parameter reference is %9 which refers to parm9. To reference parm10 the SHIFT command is used once. It will cause %0 to reference parm1, %1 to reference parm2, and so on. The new %9 will therefore reference parm10.

SHIFT can be used as many times as you like within a batch file. The only drawback to it is that each time it is used the lowest parameter is lost.

SORT

Type:	External.
Purpose:	Used to sort lines in files and the output of DOS commands
Syntax:	`SORT /R /+col <filename` `command | SORT /R /+col`
Parameters:	`/R` is an optional parameter that causes sorting to occur in reverse order. `/+col` is an optional parameter that causes sorting to be done, based on the characters starting in the column `col`. `<filename` is the name of the file whose contents will be sorted. The name must be preceded by a less-than sign. The file name may include drive and path specifiers. The output will appear on the screen. `command` is a DOS command whose output will be sorted. For example, the command `DIR	SORT` will create a directory listing that is sorted alphabetically.
Comments:	Only ASCII text files should be sorted. For characters in the first half of the ASCII table (characters 0 through 127) all sorting is done in the order in which characters appear in the ASCII table, except that the sorting is not case sensitive — the letter a is equivalent to the letter A.

For characters greater than 127 in the ASCII table the sort order is determined by the country code currently in use.

If you want the output to go somewhere other than the screen, you may specify that with a greater-than sign and the destination following the SORT command. For example:

```
SORT /R /+col <filename >outfile
```

will create a sorted copy of `filename` in `outfile`. Outfile refers to a device or file name that will receive the output. For example, if outfile is PRN: then the output will appear on the printer. If outfile is NEWFILE.TST then the output will be placed in a file named NEWFILE.TST.

UBST

ype: External.

urpose: Used to substitute a drive letter for a path. It is useful for programs that do not recognize paths but do recognize drive letters.

yntax:

```
SUBST newdrive olddrive:path
SUBST drive /D
```

arameters: `newdrive` is the drive letter that will be used to refer to the olddrive and path.

`olddrive:path` is the path being substituted with the newdrive letter. You may use just a path if the drive is the current drive.

`/D` is used to delete the substituted drive letter specified by drive.

omments: Entering SUBST with no parameters will display all current substitutions.

You may only specify drive letters less than or equal to the drive letter specified in CONFIG.SYS with the LASTDRIVE command. If LASTDRIVE is not included in CONFIG.SYS then the largest drive letter you may use is E. The new drive letter cannot be the current drive.

The substituted drive letter can be used in place of the drive and path that it is replacing.

The following commands should not be used when a substitution is in place: ASSIGN, BACKUP, DISKCOMP, DISKCOPY, FDISK, FORMAT, LABEL, and RESTORE.

You should be very careful and test the effects before using the following commands when a substitution is in place: CHDIR, MKDIR, RMDIR, and PATH.

If you have a very long subdirectory path that you reference frequently, using SUBST to substitute a drive letter for the path can save a lot of typing.

If you want to replace an entire drive letter with another drive letter, you should use the ASSIGN command. SUBST is intended for substituting drive letters for subdirectory references.

SWITCHES

Type: CONFIG.SYS

Purpose: Causes an enhanced keyboard to behave like a conventional keyboard.

Syntax: `SWITCHES=/K`

Parameters: Causes DOS to behave as though a conventional keyboard is installed when an enhanced keyboard is actually installed.

Comments: Some applications do not run properly when an enhanced keyboard is installed. SWITCHES will allow you to run these applications with an enhanced keyboard.

SYS

Type: External.

Purpose: Used to transfer the hidden system files to a disk.

Syntax: `SYS drive`

Parameters: `drive` is the drive to which the files will be transferred.

Comments: Transferring the hidden system files to a disk, as well as the COMMAND.COM file, makes the disk a bootable disk.

SYS may move files that already exist on your disk. To be safe you should back up your disk before transferring the system files.

SYS can be used on a disk that has been formatted without the /S or /B parameters, as long as the disk is blank. SYS may be used to transfer files to a disk that has been formatted with the /B parameter at any time, since /B reserves the space for the system files and does not letter other files occupy that space.

TIME

Type: Internal.

Purpose: Sets the time that is stored in the internal clock.

Syntax:

```
TIME hh:mm:ss.hs
TIME
```

Parameters:

- `hh` stands for the hour (1–12 or 1–24).
- `mm` stands for the minute (1–59).
- `ss` stands for the second (1–59).
- `hs` stands for the hundredth second (1–99).

Comments: You may specify only as much of the time as you feel is important. `Enter TIME 9` will set the time to 9:00.

If you type an incorrect time, DOS will display the message:

```
Invalid time Enter new time
```

If you press Enter at this point, the time will be left unchanged.

Entering TIME with no parameters causes DOS to display the current time.

TREE

Type: External.

Purpose: Used to display a directory, its subdirectories and the files contained in them.

Syntax:

```
TREE drive:path /F /A
```

Parameters: `drive` is the drive on which the directory exists.

`path` is the path to the subdirectory to be displayed.

`/F` causes the files as well as subdirectories to be displayed.

`/A` uses graphics characters when displaying the tree.

Comments: To show the complete directory of the entire disk, specify the root directory.

Directory trees almost always fill up more than one screen. To cause the display to pause, you may use the MORE command or press Ctrl-S.

TYPE

Type: Internal.

Purpose: Prints the contents of a file on the screen.

Syntax: `TYPE filename`

Parameters: `filename` is the name of the file to be typed on the screen. It may contain a drive and path specifier but may not contain wild card symbols.

Comments: TYPE is most useful for reviewing the contents of text files.

UNDELETE

Type: External.

Purpose: Recovers files that have been deleted accidentally.

Syntax: `UNDELETE filespec /LIST or ALL /DOS or DT`

Parameters: `filespec` is the file specifier for the file(s) that you want to undelete. A drive and/or path may be specified. Wild cards may be used.

`/LIST` lists the files that may be recoverable, but does not attempt to undelete them.

`/ALL` undeletes all files without prompting for confirmation for each one.

`/DOS` forces UNDELETE to use the information stored in the FAT and directories, rather than a deletion tracking file, even if a deletion tracking files is present.

`/DT` forces UNDELETE to use the information stored in a deletion tracking file.

Comments: UNDELETE is more likely to be successful if a deletion tracking files is present. Deletion tracking files are created using the MIRROR command.

If /DOS is not specified, UNDELETE will restore only those files found in a deletion tracking file, if one is present. If a deletion tracking file is not present, UNDELETE will attempt to restore files using information in the directory and the FAT.

This command is discussed in detail in Chapter 5.

UNFORMAT

Type: External.

Purpose: If a disk that has files on it is accidentally formatted, UNFORMAT may be able to undo the formatting, thereby restoring the original files. It can also be used to rebuild a damaged partition table.

Syntax:

```
UNFORMAT drive: /J /P

UNFORMAT drive: /U /L /TEST /P

UNFORMAT /PARTN /L
```

Parameters: `drive:` is the drive that contains the disk to be unformatted.

`/J` checks for the mirror file, and verifies that it agrees with the system information on the disk. This switch disables actual unformatting, and is used for information purposes only.

`/P` routes output messages to the printer.

`/U` forces UNFORMAT to ignore a mirror file, if one exists.

`/L` When used without the /PARTN switch, /L forces UNFORMAT to ignore the mirror file, and lists all files and subdirectories found by UNFORMAT. Without this switch, UNFORMAT lists only those subdirectories and files that are fragmented and, therefore, cannot be fully reconstructed without the use of the mirror file. When used with the /PARTN switch, /L displays the partition table of the current drive.

`/TEST` simulates an UNFORMAT on the specified disk, without actually changing anything on the disk.

`/PARTN` restores a corrupted partition table on a hard disk. This switch only works if a PARTNSAV.FIL file was created by the MIRROR /PARTN command.

Comments: If /U is used, then UNFORMAT attempts to unformat the disk using just information it can deduce from a cluster-by-cluster analysis of the disk. This is quite slow, but if the mirror file is very old, you may get more recent files restored.

Use /TEST to determine the effects of an UNFORMAT if you are not sure you want to do one. This can be combined with /P to produce a paper report that you can study.

This command is discussed in detail in Chapter 6.

VER

Type: Internal.

Purpose: Displays the version of DOS that is being used.

Syntax:

```
VER
```

Parameters: None.

Comments: The version is displayed as single digit before a period and two digits after, indicating major and minor versions.

VERIFY

Type: Internal.

Purpose: Causes DOS to verify all disk write operations.

Syntax:

```
VERIFY ON
VERIFY OFF
VERIFY
```

Parameters: `ON` causes DOS to perform a verification after every disk write operation.

`OFF` causes DOS to not verify disk write operations. The default value is OFF.

Comments: Entering `VERIFY` with no parameters displays the current status of VERIFY.

Disk write errors are rare so it is not normally necessary to have VERIFY set to on. VERIFY will significantly slow down your computer's performance, since the verification is a slow process. If you have highly critical data being written it may be worth the extra time.

VOL

Type: Internal.

Purpose: Displays the volume label and serial number (if there is one) of the specified disk.

Syntax: `VOL drive`

Parameters: `drive` is the drive that contains the disk whose label will be displayed.

Comments: If no drive is entered, the volume label of the current drive is shown.

XCOPY

Type: External.

Purpose: Used to copy groups of files, including those in subdirectories.

Syntax: `XCOPY source target /P /D:mm-dd-yy /W /S /E /V /M /A`

Parameters: `source` is the set of file(s) to copy. It may contain a drive and path specifier and wild cards.

`target` is the path to copy to. It may contain a drive and path specifier and wild cards.

`/P` causes XCOPY to prompt you with a (Y/N)? before copying each file.

`/D:mm-dd-yy` copies files whose creation or last modification date is on or after the date specified. The date format will depend on the country code, that shown is for the U.S.

`/W` causes XCOPY to wait while you insert disks into the source drive.

`/S` specifies that subdirectories should be copied in addition to the specified or current directory. If /S is not included, only the files in the specified or current directory are copied. If /S is copied, any subdirectories that do no exist on the target will be created, as long as they are not empty subdirectories on the source.

`/E` tells XCOPY to create subdirectories on the target, even if they are empty.

`/V` turns on write verification for the XCOPY operation. This will slow down the copy process considerably due to the extra work involved in verifying that each file is written properly to the disk. This is not usually necessary, since write errors are rare. However, if your data is highly critical you may want to use it as a safeguard.

`/M` is used to copy only those files that have the archive attribute set to on. These are files that have been created or modified since the last BACKUP command or XCOPY /M command. This is useful if XCOPY is used for backing up files. When /M is specified XCOPY sets the archive attribute to off after the copy is completed.

`/A` is used to copy only those files that have the archive attribute set to on. These are files that have been created or modified since the last BACKUP command or XCOPY /M command. This is useful if XCOPY is used for backing up files. When /A is specified XCOPY does not change the archive attribute of the file.

Comments: If no drive is specified, XCOPY starts with the current directory. If no file name is specified, XCOPY assumes *.*. XCOPY does not copy hidden files.

You can rename files with the XCOPY command in the same ways you can rename them with the COPY command.

XCOPY cannot copy a file larger than the target disk. If you need to do this, you should use the BACKUP command. However, files created with XCOPY are standard DOS files so they can be used without a RESTORE operation.

Appendix

IBM Extended ASCII Character Set

Decimal	Hex	Binary	Meaning	Control Key	Character
0	0	00000000	NUL (null)	^@	
1	1	00000001	SOH (start-of-header)	^A	☺
2	2	00000010	STX (start-of-transmission)	^B	☻
3	3	00000011	ETX (end-of-transmission)	^C	♥
4	4	00000100	EOT (end-of-text)	^D	♦
5	5	00000101	ENQ (enquiry)	^E	♣
6	6	00000110	ACK (acknowledge)	^F	♠
7	7	00000111	BEL (bell)	^G	•
8	8	00001000	BS	^H	◘
9	9	00001001	HT (horizontal tab)	^I	○
10	A	00001010	LF (line feed)	^J	◙
11	B	00001011	VT (vertical tab)	^K	♂
12	C	00001100	FF (form feed)	^L	♀
13	D	00001101	CR (carriage return)	^M	♪
14	E	00001110	SO (shift out)	^N	♫
15	F	00001111	SI (shift in)	^O	☼
16	10	00010000	DLB (data link escape)	^P	►
17	11	00010001	DC1 (X-ON)	^Q	◄
18	12	00010010	DC2 (tape)	^R	↕
19	13	00010011	DC3 (S-OFF)	^S	‼
20	14	00010100	DC4 (no tape)	^T	¶
21	15	00010101	NAK (negative acknowledge)	^U	§
22	16	00010110	SYN (synchronize)	^V	▬
23	17	00010111	EBT (end-of-transmission block)	^W	↨
24	18	00011000	CAN (cancel)	^X	↑
25	19	00011001	EM (end of medium)	^Y	↓
26	1A	00011010	SUB (substitute)	^Z	→
27	1B	00011100	ESC (escape)	^[	←
28	1C	00011101	FS (file separator)	^\	∟
29	1D	00011101	GS (group separator)	^]	↔
30	1E	00011110	RS (record separator)	^^	▲
31	1F	00011111	US (unit separator)	^_	▼

Decimal	Hexadecimal	Binary	Character
32	20	00100000	
33	21	00100001	!
34	22	00100010	"
35	23	00100011	#
36	24	00100100	$
37	25	00100101	%
38	26	00100110	&
39	27	00100111	'
40	28	00101000	(
41	29	00101001	)
42	2A	00101010	*
43	2B	00101011	+
44	2C	00101100	,
45	2D	00101101	-
46	2E	00101110	.
47	2F	00101111	/
48	30	00110000	0
49	31	00110001	1
50	32	00110010	2
51	33	00110011	3
52	34	00110100	4
53	35	00110101	5
54	36	00110110	6
55	37	00110111	7
56	38	00111000	8
57	39	00111001	9
58	3A	00111010	:
59	3B	00111011	;
60	3C	00111100	<
61	3D	00111101	=
62	3E	00111110	>
63	3F	00111111	?
64	40	01000000	@
65	41	01000001	A
66	42	01000010	B
67	43	01000011	C
68	44	01000100	D
69	45	01000101	E

continued...

…from previous page

Decimal	Hexadecimal	Binary	Character
70	46	01000110	F
71	47	01000111	G
72	48	01001000	H
73	49	01001001	I
74	4A	01001010	J
75	4B	01001011	K
76	4C	01001100	L
77	4D	01001101	M
78	4E	01001110	N
79	4F	01001111	O
80	50	01010000	P
81	51	01010001	Q
82	52	01010010	R
83	53	01010011	S
84	54	01010100	T
85	55	01010101	U
86	56	01010110	V
87	57	01010111	W
88	58	01011000	X
89	59	01011001	Y
90	5A	01011010	Z
91	5B	01011011	[
92	5C	01011100	\
93	5D	01011101	]
94	5E	01011110	^
95	5F	01011111	_
96	60	01100000	`
97	61	01100001	a
98	62	01100010	b
99	63	01100011	c
100	64	01100100	d
101	65	01100101	e
102	66	01100110	f
103	67	01100111	g
104	68	01101000	h
105	69	01101001	i

continued…

...from previous page

Decimal	Hexadecimal	Binary	Character
106	6A	01101010	j
107	6B	01101011	k
108	6C	01101100	l
109	6D	01101101	m
110	6E	01101110	n
111	6F	01101111	o
112	70	01110000	p
113	71	01110001	q
114	72	01110010	r
115	73	01110011	s
116	74	01110100	t
117	75	01110101	u
118	76	01110110	v
119	77	01110111	w
120	78	01111000	x
121	79	01111001	y
122	7A	01111010	z
123	7B	01111011	{
124	7C	01111100	¦
125	7D	01111101	}
126	7E	01111110	~
127	7F	01111111	
128	80	10000000	Ç
129	81	10000001	ü
130	82	10000010	é
131	83	10000011	â
132	84	10000100	ä
133	85	10000101	à
134	86	10000110	å
135	87	10000111	ç
136	88	10001000	ê
137	89	10001001	ë
138	8A	10001010	è
139	8B	10001011	ï
140	8C	10001100	î
141	8D	10001101	ì

continued...

...from previous page

Decimal	Hexadecimal	Binary	Character
142	8E	10001110	Ä
143	8F	10001111	Å
144	90	10010000	É
145	91	10010001	æ
146	92	10010010	Æ
147	93	10010011	ô
148	94	10010100	ö
149	95	10010101	ò
150	96	10010110	û
151	97	10010111	ù
152	98	10011000	ÿ
153	99	10011001	Ö
154	9A	10011010	Ü
155	9B	10011011	¢
156	9C	10011100	£
157	9D	10011101	¥
158	9E	10011110	₧
159	9F	10011111	ƒ
160	A0	10100000	á
161	A1	10100001	í
162	A2	10100010	ó
163	A3	10100011	ú
164	A4	10100100	ñ
165	A5	10100101	Ñ
166	A6	10100110	ª
167	A7	10100111	º
168	A8	10101000	¿
169	A9	10101001	⌐
170	AA	10101010	¬
171	AB	10101011	½
172	AC	10101100	¼
173	AD	10101101	¡
174	AE	10101110	«
175	AF	10101111	»
176	B0	10110000	░
177	B1	10110001	▒

continued...

…from previous page

Decimal	Hexadecimal	Binary	Character
178	B2	10110010	▓
179	B3	10110011	│
180	B4	10110100	┤
181	B5	10110101	╡
182	B6	10110110	╢
183	B7	10110111	╖
184	B8	10111000	╕
185	B9	10111001	╣
186	BA	10111010	║
187	BB	10111011	╗
188	BC	10111100	╝
189	BD	10111101	╜
190	BE	10111110	╛
191	BF	10111111	┐
192	C0	11000000	└
193	C1	11000001	┴
194	C2	11000010	┬
195	C3	11000011	├
196	C4	11000100	─
197	C5	11000101	┼
198	C6	11000110	╞
199	C7	11000111	╟
200	C8	11001000	╚
201	C9	11001001	╔
202	CA	11001010	╩
203	CB	11001011	╦
204	CC	11001100	╠
205	CD	11001101	═
206	CE	11001110	╬
207	CF	11001111	╧
208	D0	11010000	╨
209	D1	11010001	╤
210	D2	11010010	╥
211	D3	11010011	╙
212	D4	11010100	╘
213	D5	11010101	╒

continued…

…from previous page

Decimal	Hexadecimal	Binary	Character
214	D6	11010110	╓
215	D7	11010111	╫
216	D8	11011000	╪
217	D9	11011001	┘
218	DA	11011010	┌
219	DB	11011011	█
220	DC	11011100	▄
221	DD	11011101	▌
222	DE	11011110	▐
223	DF	11011111	▀
224	E0	11100000	α
225	E1	11100001	ß
226	E2	11100010	Γ
227	E3	11100011	π
228	E4	11100100	Σ
229	E5	11100101	σ
230	E6	11100110	µ
231	E7	11100111	τ
232	E8	11101000	Φ
233	E9	11101001	Θ
234	EA	11101010	Ω
235	EB	11101011	δ
236	EC	11101100	∞
237	ED	11101101	φ
238	EE	11101110	ϵ
239	EF	11101111	∩
240	F0	11110000	≡
241	F1	11110001	±
242	F2	11110010	≥
243	F3	11110011	≤
244	F4	11110100	⌠
245	F5	11110101	⌡
246	F6	11110110	÷
247	F7	11110111	≈
248	F8	11111000	°
249	F9	11111001	·

continued…

...from previous page

Decimal	Hexadecimal	Binary	Character
250	FA	11111010	·
251	FB	11111011	√
252	FC	11111100	ⁿ
253	FD	11111101	²
254	FE	11111110	■
255	FF	11111111	

Index

The letter f preceding a page number indicates that the page contains a figure for the subject.

The letter t preceding a page number indicates that the page contains a table for the subject.

Symbols

- : (see colon)
- $ (see Dollar sign)
- % 351, 436
- %variable (FOR command) 178
- * wild card (see asterisk)
- --- More --- (command line) 165
- .. (see ellipsis) 287
- < (Less than symbol)
  - including in DOS prompt 184
  - redirection symbol 166, 168
- <DIR> 279
- > (Greater-than symbol) 288
  - as part of standard command prompt 150
  - including in command prompt 183
  - redirection symbol 166, 168
  - (see also Redirection)
  - TYPE command 320
- >> redirection symbol 167
- ? wild card (see Question mark)
- @ 444
- @ symbol in DOSHELP.HLP 142, 163
- [+] symbol in directory tree 45
- [–] symbol in directory tree 45
- [(in Shell directory tree) 45
- \ 47
-] (in Shell directory tree) 45
- ^m 98, 140
- ^Z
  - as end-of-file marker 297
  - explained 160
  - inserting in command line 161
- | (Pipe symbol) 165, 168

Numerical

- 16-bit 5
- 16-bit processing 8
- 286 9, 219, 242, 247, 373
- 32-bit processing 10
- 32-bit word 5
- 386 9, 219, 231, 232, 234, 250, 263, 373
- 8-bit processing 8
- 80286 8, 9, 219, 231, 232, 233, 234, 249, 262
  - address size 18, 228
  - expanded memory 243
- 80386 8, 9, 10, 219, 232, 234, 244, 249
  - address size 18, 228
  - expanded memory 230, 243
- 80386DX 10
- 80386SX 10
- 80486 8, 10, 219
  - address size 18
- 8086 8, 10
- 8088 8, 9, 10, 228, 231, 250
  - address size 228
  - expanded memory 243
- 8088/8086 219, 234, 262
- 80x86 family 9, 10
  - addressing scheme 233
- 80x87 9
- 8514/A video card 12

A

- A20 handler 258, 260
  - codes 259
- Active Task List 93, f110, 112

  adding programs 111
  in Program/File List view f110
  managing 111
  terminating programs 113
Add Group dialog box 102, 104
Add Program dialog box 88, 97, 104
  protected by passwords 94
Addressing scheme 262
Advanced Color Adjustments 144
Advanced command button 35
Advanced dialog box 97, 98, 99
Advanced Options dialog box
  for Shell Programs f97
  Showing End of Help Text for Editor f141
Advanced Properties dialog box
  for Editor program 140
All Files View 29
Allocation units 391
Alphabetic characters
  internal representation 5
American Standard Code for Information Interchange 64
  (see also ASCII)
ANSI.SYS 191, 459, 460
  adjusting DOS prompt color 190, 191
  customizing screen and keyboard 460-472
  loading into a UMB 236
  PROMPT command 460-473
  size 237
APPEND command 395,489
  /E switch 396
  /X switch 351, 396
Application Shortcut Key text box in Shell 92
Archive file attribute 50, 78, 79, 356
  BACKUP command 312
  described 79
  XCOPY command 312
Arrow keys
  in active Shell area 26
  and Shell menus 33
ASCII 64, 269
  codes 66
  creating from command stack 198
  defined 59
  files 59
  Filter commands 169
  table 266
  SORT command sorting 174
ASCII files
  compared to binary 297
  COPY command 297
ASCII option on Shell Display menu 66
ASSIGN command 397, 489
  compared to SUBST command 400
Associate File dialog box f57, 59
Associate option 52, 57, 58
Asterisk (*)
  expanding directory tree 45
  wild card 46, 272
ATTRIB command 316, 350, 356, 491
Attributes 50, 51
  changing in Shell 77
  defined 354
  dialog box for changing f77
  dialog box for confirming f78
  directory 119
  in Show Info dialog box 50
  four types shown in Shell 78
  settings 77
AUTOEXEC.BAT 456
  as ASCII file 64
  PRINT command 60
  LOADHIGH command 238
  PATH command 153
  PROMPT command 182
  environment variables 213
AUX 268

B

Back option on Shell Help screens 37
Back-filling memory 234, 235
Background colors 144
Backslash 40, 47, 274, 287, 312
Backspace in DOS prompt 189
BACKUP command 320,492
  archive attribute 312
  log file 323
Backward compatible 9
Base two (see binary number system)
BAT file extension 84
BATCH 269
Batch file command processor 104
Batch files 431-453
  creating from command stack 198
  executing from Shell programs 105
  versus Shell programs 432
  writing 433-450
Binary 221, 222, 266
  digit 5
  files vs. ASCII files 297
  COPY command 297
  number system 5
Binary files, COPY 297

BIOS (see ROM-BIOS)
Bits 8, 266
  defined 5, 220
Boot Record 368, f369
  location 372
Booting 368
Brackets 87
  as group indicators in Program List 102
  in Shell directory tree 45
  used to identify Shell groups 87
BREAK command 485, 494
Buffers
  default number t402
  DOSKEY 210
  loading into HMA 404
  look-ahead 403
  recommended settings for various hard disks t403
BUFFERS command 401, 404, 495
Buttons on Shell dialog boxes
  Advanced 35
  Back 37
  Cancel 35
  Close 37
  Help 35, 37-38
  Index 37-38
  Keys 37-38
  OK 35
  Option 34-35
Bus 8, 10, 262
Bytes 8, 221
  defined 5, 220, 266

C

CAD 14
CALL command 451, 452, 453, 496
  in Shell programs 105
Cancel command button 35
Caret in Shell help definitions 98, 140
Carriage return in Shell help 98
CD command (see CHDIR command)
Central Point Software 84
Central Processing Unit 8, 217
CGA 10, 11, 12, 115
  problems in Shell programs 101
Change Attributes option 52, 77
Change option 426
CHDIR command 275, 346, 498
Check boxes
  using 34
Chips 219, 228, 373
  defined 8
  ROM 220
  types used for CPU 8
  (see also 8088, 8086, 80286, 80386, 80486, 80586)
CHKDSK command 342, 391, 498
  redirecting output 393
Clock speed 9
CLOCK$ 268
Close option on Shell Help screens 37
CLS command 176, 500
Clusters 301, 302, 330, 331, 332, 335, 338, 339, 342, 354, 355, 357
Collapse Branch option 45
Collating sequence
  for SORT command 174
Colon 274
  as part of drive identifier 16, 40, 150
  label indicator in Batch files 445
Color Graphics Adapter (see CGA)
Color schemes for MS-DOS Shell 114, 115
Colors
  adjusting in Editor 427
  adjusting with PROMPT 465
  available for Shell screen 147
  changing in the Shell 114, 115, 116
  creating custom Shell screens 144
  in DOSSHELL.INI 139
  setting for DOS prompt 190
Colors option on Shell Options menu 115
COM file extension 66, 84
Command buttons described 35
Command interpreter 458, 459
Command line 21, 149-217
  canceling 161
  copying to the template 161
  defined 150
  deleting single characters 161
  editing keys t161, 162
  explained 151
  leading blanks 156
  size limit 155
  turning off line wrap 469
  using 155-176
Command prompt 60
  customizing 182-191
  default setting 150
  Doskey program 191
  explained 150
  moving files 68
  option on Shell Program List menu 30

  program in Program List 87
  reasons for using 30
  returning to Shell 30
COMMAND.COM 30, 31, 151, 214, 458
  size 224
Commands text box in Shell 89, 104
Comments
  adding to DOSSHELL.INI 138
  in DOSHELP.HLP file 163
COMP command 290, 291, 500
  compared to FC 304
  switches for 292
Computers 5
  described 5
  memory design 18
  use of binary addresses 222
COMSPEC 457
COMx 268
CON 268
CONFIG.SYS 190, 191, 244, 245, 456, 458, 459
  BUFFERS command 226
  DEVICEHIGH command 236
  DOS command 235
  EMM386 245
  FILES command 225
  INSTALL command 243
  installing HIMEM.SYS 246
  setting environment size 214
  setting up memory managers 235
Configuration commands 455
Confirm Mouse Operation dialog box 118
Confirm on Mouse Operation option 118
Confirmation dialog box in Shell 44
Confirmation Dialog Boxes, Controlling 116
Confirmation option 117
Contiguous memory 234
Control-break 439, 485
Control-Z symbol (see ^Z)
Conventional Memory text box 99
Copy command 62, 293, 502
  alternative to Shell File Print option 62
  disadvantages 63
  DISKCOPY, compared 301
  switches 296
  to change DATE/TIME stamps 298
  wild cards 300
Copy option in Editor Edit menu 424
Copy option on Shell File menu 52
Copying files in Shell
  with keyboard 73
  with mouse 72
COUNTRY command 460, 505
  effect on SORT command 174
COUNTRY.SYS file 460
  SORT command 174
CPU 8, 9, 19, 217, 218, 219
  chips 8
  clock speed 9
  primary functions of 8
  protected mode 228
  real mode 228
Create Directory Dialog box 43
Create Directory option 43
Current drive 150
"Current drive is no longer valid" 186
Cursor 26
  moving in the Editor 422
Cut option in Editor Edit menu 424
Cylinders 363

D

Data Area of disks 368
Data files 16
  compared to program files 269
Date (including in DOS prompt) 186
DATE command 506
Date stamp
  changing with COPY 298
Default Parameters text box 91
DEL command 176, 300, 301, 318, 507
Delete File Confirmation 74, 75, 117
Delete Item dialog box 107
Delete option, Shell File Menu 52, 74
  disabled if root directory selected 44
"Delete Tracking file not found" 82
Deleting directories 43, 44
  from within the Shell 117
Deleting files in Shell 74
Deletion tracking file 336, 337
Descending order check box 48
Deselect All option 52, 55
DESQview 393
DEVICE command 508
Device drivers 238, 459
  loading into UMBs 236
  memory requirements 237, 238
  memory used 226
DEVICEHIGH command 235, 236, 242, 508
  risk of locking up PC 236
  size parameter 236, 238

Dialog boxes 143
  adding to Shell programs 86
  defined 33
  described 38
  for programs in Program List area 85
  linking in DOSSHELL.HLP file f143
  using 31-39
  (see also using dialog box titles)
Diamond indicator on menus 32
DIR command 176, 355, 509
  Hidden attribute 332
  sorting output 282
  switches 177, 280
DIR program in MS-DOS Shell 89, 90, 91
DIR program (DOSHELL.INI) f95
DIRCMD environment variable 163, 212, 213, 456, 457
Direct Memory Access 256
Directories 17, 281
  cannot be undeleted 80
  creating in the Shell 43
  described 39, 265
  managing in the Shell 39-51
  naming 274
  organizing 289
  renaming 44
  root (see root directory) 40
  rules for naming 43
  selecting 42
  sorting 282
  special types of files as 119
  synonymous with "subdirectories" 40
  understanding and managing 217-263
Directory attribute 119
Directory tree 17, 41, 273, 274, 275, 277, 287
  All Files View 29
  Dual File List View 29
  keeping current 29
  Single File List View 29
Directory Tree area of Shell 25
Disk cache 373
Disk drives 14, 362, 367
  letter identifiers 16
  upgrading 7
Disk labels 278
Disk Operating System defined 19
Disk partitions 386
Disk space 391
  as virtual memory 228
Disk Utilities group 79, 87
DISKCOMP command 394, 510
DISKCOPY command 301, 394, 511
Disks
  clusters (see Clusters)
  compared to memory 14
  fragmentation 384, 413
  logical structure 363-372
  managing 372-414
  partitions (see Partition)
  tracks 363, 367
  understanding and managing 359-414
  (see also Floppy disks)
  (see also Hard disks)
Display hidden/system files check box 48
Display option 427
DISPLAY.SYS 459
DMA 256, 257
Dollar sign
  adding to command Prompt 183
  in macros 209
  in PROMPT command parameters 182
DOS 232
  command line (see Command line)
  configuring and customizing 455-485
  defined 19
  external commands 20, 224
  internal commands 20, 224, 290
  memory requirements 20
  memory restrictions 18
DOS 5.0 memory managers 231, 234, 235, 244
DOS 5.0 Shell 23-148
DOS command 235, 249, 512
  HIGH and UMB switches 247
DOS commands
  commonly used commands 176-182
  error messages 167
  syntax 153
  (see also individual commands)
DOS environment 456-459
  copy loaded with TSRs 243
  described 212
DOS extenders 231
DOS filters (see Filter Commands)
DOS operating system 1
DOS prompt 150, 456
  resetting to default value 182
  Using ANSI.SYS for Cursor Positioning f462
DOSHELP.HLP file 163
DOSKEY command 177, 192, 513
  /BUFSIZE switch 216
  /REINSTALL switch 216
  running more than once 210
  syntax 210

- Doskey program 191
  - buffer 210
  - command stack 191-200
  - default memory usage 193
  - described 191
  - editing keys 197
  - explained 177
  - FOR command 178
  - FIND command 178
  - how it works 215
  - insert mode 197
  - loading into UMB 238
  - macro features 200-211
  - memory management 210, 215
  - multiple commands per line 180, 192
  - overstrike mode 198
  - paragraph mark 180
  - redirection keys t197
- DOSSHELL command 31, 119, 513
  - different from SHELL 214
  - switches for 118
- DOSSHELL.HLP 139, 140, 141, 142, 143, 144
- DOSSHELL.INI 25, 95, 96, 98, 121, 137, 138, 140
  - as ASCII file 137
  - comments in 138
  - copying between computers 139
  - creating customer color schemes 144
  - customizing 137
  - DIR Program section f95
  - help screens, adding 98
  - locating programs 96
  - passwords 95
  - sample listing 122
  - storing switch settings 138
  - structure 121
  - two main sections 138
  - when updated on disk 138
- Double period (see ellipsis) 287
- DRAM 261
- Drive
  - changing current drive 150
- Drive Icon area of Shell 25
- Drive letter
  - including in DOS prompt 84
- DRIVER.SYS 459
- Dual File Lists View 29
- Dynamic RAM 261

E

- ECHO command 513
  - in Shell programs 104
  - REMARK command 443
- EDIT command 155, 416, 429
- EDIT.COM file 429
- EDIT.HLP 428
- Editor program in Program List 87
- EDLIN 415
  - ASCII files 59
- EGA 12, 115
- EGA.SYS 459
- Electronic memory (see Memory)
- Ellipsis
  - following command buttons 35
  - on Shell menus 32
  - parent directory specifier 287, 288
- EMM386 231, 232, 235, 240, 242, 244, 245, 249, 251, 253, 254, 255, 373
  - /NOEMS switch 242
  - /RAM switch 242
  - cannot be loaded into UMB 238
  - handles 256
  - switches 253
  - /W switch 245
  - Windows 246
- EMM386.EXE 459
- EMS 229, 254, 255, 256
- Enable Task Swapper option 109
- End of file marker 160
  - (see ^Z)
- Enhanced Graphics Adapter (see EGA)
- Enhanced Keyboard 14
- Enhanced mode 246, 263
- Entering Multiple Commands on One Line 192
- Environment described 212
  - (see also DOS environment)
- Environment variables 456, 457
  - described 212
  - PATH 437
  - referencing in batch files 437
- Equal Sign (in DOS prompt) 188
- ERASE command 301
- Error messages for DOS commands 167
- ERRORLEVEL environment variable 304, 316, 379, 394
  - RESTORE values 330
  - XCOPY values 316
- Escape sequences 460
  - PROMPT command 464
- EXE file extension 66, 84
- EXE2BIN.EXE 373
- Executable files 16, 66, 270, 350
  - defined 17

Exit codes (see ERRORLEVEL environment variable)
EXIT command 30
Exit option on Shell File menu 31, 52, 87
Exiting the Shell 30
  how exiting works 30
  temporarily 30
EXPAND command 514
Expand All option 45
Expand Branch option 45
Expand One Level option 45
Expanded Memory 228, 229, 242, 245, 249, 252, 254, 255, 256
  advantages and disadvantages 230
  converting extended memory 231, 232, 243
  defined 233
  disk cache and 373
  Windows and 232, 246
Expanded memory emulator 244
Expanding the directory tree 45
Expansion slots 7, 220
  sizes 7
Extended DOS partition 386
Extended Graphics Array 12
Extended memory 100, 231, 232, 244, 245, 246, 249, 255, 258, 260
  advantages and disadvantages 250
  amount used by UMBs 241
  converting to expanded memory 231, 232, 243, 245
  defined 224, 231, 233
  disk cache 373
  effect on Task Swapper 100
  HIMEM.SYS 247
  not for 8088 PCs 231, 246
  reduced by back-filling UMBs 241
  shadow RAM 260
  specifying for Shell programs 100
  used for back-filling UMBs 234
  Windows and 232
Extended Memory Specification 231
External commands (see DOS external commands) 290

F

FASTOPEN command 405, 515
FAT 330, 331, 332, 333, 340, 342, 354, 355, 358, 368
FC command 304, 516
  compared to COMP 304
  syntax 305
FDISK command 386
  Menu f387
FFIND macro 178-179
File Allocation Table (see FAT)
File Display Options 46, 56
File Display Options dialog box f46
File handles 234, 235
File List area of Shell 25
  changing display 46
  display of hidden/system files 48
  Dual File List View 29
  keeping current 29
  running programs 85
  Single File List View 29
  title display 48
File List Help Screen 37
File Menu in Shell f32
  options dependent on active area 31
  Program List 87
File Options dialog box 117
File names
  as program parameters 272
  extensions 84, 268, t271
  legal characters 267
Files
  attributes (see Attributes)
  data 268, 270
  handles 225
  managing in the Shell 39
  opening 225
  program 268
  types 16
  understanding and managing 265-358
FILES command 225, 517
Filter Commands 168, 169
  switch settings t169
FIND command 168, 173, 517
  Doskey macros 206
  FOR command 178
  piping output 172
  terminating 171
Find option in Editor Search menu 425
Find What text box in Editor 425
Finding Text in Editor 425
Firmware 5
Floppy disks 15, 16, 273, 361
  sizes 15, t16
FOR command 178, 433, 446, 447, 448, 449
  %variable use 178
  FIND command 171, 178
  limited to one per command line 179
  output piping 179

- output redirection 179
- searching for multiple text strings 179

Foreground colors
- for Shell screens 144

Format 368, 374

FORMAT command 180, 370, 374, 518

Forward compatible 9

Fragmentation (see disks) 384

Full path name 152

G

GOTO command 433, 444, 446, 520

GRAPHICS command 521

Graphics adapter cards
- common types 115
- graphics mode Shell display 115

Graphics mode 101
- differences from text mode 116
- in MS-DOS Shell 24, 115, 116

Greater-than sign 182
- as standard prompt terminator 184
- in command prompt 150
- including in DOS prompt 183

Group Properties dialog box 104

Groups in Shell
- adding to Program List 88, 101, 102
- changing help text 104
- changing titles 104
- copying programs between 105, 106
- defined 101
- deleting 107
- distinguishing from programs 102
- how they are identified 87
- nesting 102
- organizational use in Program List 101

H

Hard disks 15, 16, 17, 273, 362, 363, 367
- advantages 16
- manufacturing 365
- sizes 16

Hardware 5, 7

Help 163
- accessing from Program List 99
- adding hypertext (see hypertext)
- adding to command line help 164
- adding to Shell programs 86, 97
- context-sensitive 37
- exceeding 256 character limit 98
- for programs in Program List area 85
- hypertext feature (see hypertext)
- in DOSSHELL.INI 139
- in Editor 428
- on command line 162, 163
- within the Shell 36

HELP command 162, 522
- compared to /? switch 163

Help command button 35

Help dialog boxes
- for Shell groups 87
- for Shell programs 87

Help files in Editor 428

Help option
- on Shell dialog boxes 38
- on Shell Help screens 38

Help Path option 428

Help Text box in Shell 98
- when adding groups 102

Hercules video card 12

Hex mode of Shell View screen 67

Hex option on Shell Display menu 66

Hexadecimal number system 67

Hidden attribute 354

Hidden files 354
- attribute 78, 332
- described 78
- in File List 48

Hidden files check box 48

High Memory Area (see HMA)

HIMEM.SYS 232, 235, 240, 244, 246, 249, 257, 459
- /A20 switch 260
- /CPUCLOCK switch 261
- /MACHINE switch 258
- /SHADOWRAM switch 260
- A20 handler and 258, 260
- cannot be loaded into UMB 238
- INT15 switch 258
- switches 247

HMA 247, 248, 249, 250, 257, 258
- loading BUFFERS into 404

Hypertext in Shell Help system 139, 143
- creating 139
- defined 37

I

IF command 444, 522
- in Shell programs 104

Including $ in the Prompt 183

Index option on Shell Help screens 38

Input redirection 168
Insert mode
  Doskey program 197
  on command line 158, 161
INSTALL command 243, 406, 523
Integrated circuits (see chips)
Interleave 366, 368, 414
Internal character set 374
Internal clock 9
Internal commands 151
  (see DOS internal commands) 290
International standards support 374
IO.SYS 48, 79, 355

J

JOIN command 399, 523
  incompatibilities 400

K

K (abbreviation for Kilobyte) 6
KB (abbreviation for Kilobyte) 6
Key-released code 13
KEYB command 14, 524
Keyboard 13
KEYBOARD command 460
KEYBOARD.SYS 460
Keys
  icons in this books 4
Keys option on Shell Help screens 38
Kilobytes 255
  defined 5, 221, 266

L

LABEL command 278, 355, 389, 390, 526
Labels
  in Batch files 444
Laptop computers 14
LASTDRIVE command 401, 484, 526
Less-than sign
  including in DOS prompt 184
LIM 229, 254, 255, 256
List boxes
  described 35
LOADHIGH command 235, 238, 239, 242, 406, 527
Log file 323
Logical Drives 484
Look-ahead buffers 403
Lost allocation units 391, 392
Lotus Intel Microsoft Expanded Memory Specification 229
Low-level format 374
LPT 160, 168, 268
  specifying in PRINT command 478

M

M (abbreviation for Megabyte) 6
Macros 191, 201
  creating 200
  creating a master list 209
  leading blanks 156, 202
  listing, deleting, and saving 206
  redirecting I/O 205
  running 201
  saving in sorted order 208
  using replaceable parameters 203
  using same name as a DOS command 156
  with multiple commands 202
Mapping (see Memory mapping)
Match Upper/Lowercase check box in Editor 425
MCGA 12
MD command (see MKDIR) 349
MDA 10, 11
Meg (abbreviation for Megabyte) 6
Megabytes 5, 6
  defined 221, 266
Megahertz 9
MEM command 238, 528
  /CLASSIFY switch 239, 248
  Upper Memory display 240
Memory 18, 19, 217-264
  640K barrier 222
  ability to run in UMB 238
  addresses 18
  absolute 251
  amount for Shell program Video 101
  back-filling 234, 235
  capacity of Personal Computers 218
  compared to disks 14
  contiguous 234
  conventional 233, 249
  displaying contents 237
  effect of using UMBs 239, 240
  expanded 243
  extended 246
  first meg 224
  in 1 meg computer 224
  in 640K computer 223
  increasing 7
  layout of first megabyte 233

  limits on size 18, 222
  loading programs into UMBs 238
  managing 233-250
  minimum for Shell program 100
  non-volatile 220
  RAM (see RAM)
  reserved 222, 224
  ROM (see ROM)
  segmented 250
  specifying requirements for Shell programs 99
  three types 233
  understanding 219-231
  units 18, 221
  used by BUFFERS 226
  used by DOS environment 225
  used for file handles 225
Memory managers 230, 231, 232, 234, 242, 249
  provided with DOS 5.0 244
Memory specification text boxes 99
Menu Bar
  area of Shell 25
  for View screen 65
  toggling on and off 36
  using 31
Menus
  options not available 32
  using 31-39
  switches 32
MGA 12
MHZ 9
Microsoft C 456
MIRROR command 332, 530
MIRROR program 80, 81
MIRROR.BAK file 333
MIRROR.FIL file 332
MKDIR command 349, 530
MODE command 13, 14, 473
Modes
  screen 466
Monitor 10, 13
  components 10
  upgrading 7
Monochrome Display Adapter 10
Monochrome monitor 10
MORE command 165, 168, 531
  Doskey macros 206
  FIND command 172
  (see More filter) 319
MORE filter 319
  FC command 310
  may create a temporary file 335
  TREE command 277
Motherboard 7
Mouse 14
  loading driver in UMB 238
  moving files 69
  use on Shell menus 33
Mouse pointer
  in graphics mode 26
  in text mode 26
Move operation explained 120, 121, 358
Move option on Shell File menu 52
MS-DOS 21
MS-DOS Editor 140, 415-430
  ASCII files 59
  editing keys t422
  requires QBASIC.EXE 429
MS-DOS QBasic
  program in Program List 87
MS-DOS Shell
  appearance at boot time 24
  areas of 25
  screen f24
  starting 24, 118
MSDOS.SYS 48, 79, 355
Multi Color Graphics Array 12
Multi-tasking 256
Multiple commands per line 192
Multiple lines in DOS prompt 189
Multiple programs (running in Shell) 99

N

New name text box in Shell 76
New Program Object dialog box 88
Non-DOS partitions 386
Non-volatile memory 220
NUL 268
Numbers, real 6

O

OK command button 35
Open option on Shell File menu 52, 59, 85, 87
Opening a file 57
Operating system 48
  alternatives to DOS 20
  memory limits 218
  scan codes 14
Optical disk 360
Option buttons
  mutual exclusivity 34

  selecting 35
  using 34
Options menu 56
  Colors option 115
  confirmation option 117
OS/2 220
Output redirection 167
  (see also >)
Overstrike mode
  Doskey program 198
  for Doskey program 211

P

Page frame 229, 235, 242, 244, 251, 253
  size 252
Paragraph mark 192
  as command separator 180
Parameter dialog boxes
  creating 89
  defining 87
Parameters
  defined 436
  for DOS commands 154
  for macros 203
  in batch files 452
Parent directory defined 288
Partition 371
  table 372, 385, 386
  types 386
PARTNSAV.FIL 385
Passwords
  adding to Shell programs 93
  advisability of using 94
  creating hierarchy 103
  dialog box f94
  entering 93
  for groups 103
  for Shell groups 87
  for Shell programs 87
  recovering 94
  text box in Shell 93
Paste option in Editor Edit menu 424
Path 351, 352, 456
  APPEND command 351
  clearing 351
  environment variable 437
  including in DOS prompt 185
  SUBST command 351
  understanding 151-153
  viewing 152
PATH command 152, 350, 532
  in AUTOEXEC.BAT file 153
Pause after exit check box 93
PAUSE command 444, 533
PC TOOLS 84
PC-DOS 21
PCTRACKR.DEL file 333, 334
  default sizes t334
Period
  current directory specifier 276, 287
  part of file name 268, 274
  part of file specifier 17
Peripheral devices 219
Personal Computers 1, 13, 219
  memory capacity 218
  organization 7
  types of chips 8
  video standards 11
Pipe 165, 168
  Doskey macros 206
  include symbol in DOS prompt 188
  using 168
Pixels 10, 11, 13, 467
  defined 10
Ports 475
  described 60
  printer 168
Prevent Program Switch check box 101
Primary DOS partition 386
Print buffer 477
PRINT command 60, 62, 476, 533
  /D switch 475
  /M switch 479
  /Q switch 477
  /S switch 478
  /T switch 480
  /U switch 479
  in AUTOEXEC.BAT file 60
  must be executed at DOS prompt 60
  wild cards 483
Print option on Editor File menu 421
Print option on Shell File menu 52, 59
  alternative 62
  with non-ASCII files 59
Print queue 60, 473-483
  buffer 60
  viewing contents 62
PRINTER.SYS 459
PRN 268, 288
  COPY command 62, 299
Program files
  compared to data files 269

Program Information text box 90
Program Item Properties dialog box 90, 97
Program List 85, 87, 102
  advantages of using 97
Program List area of Shell 25, 86-108
  described 87
  File menu 87
  groups 101
  pre-installed DOS programs 85
  running programs from 86
Program List menu
  Adding a group to f102
  Command Prompt option 30
Program List option 86
Program List View 29, f86
Program/File Lists View 29
Programs 139
  adding to a group 103
  adding to Active Task List 111
  adding to Program List 88
  copying between groups 105, 106
  deleting from groups 107
  distinguishing from groups 102
  in Program List area 85
  locking up under Task Swapper 113
  running from the Shell 84, 85
  running with Task Swapper 108
  terminating from Active Task List 113
Programstarter (DOSSHELL.INI) 138
PROMPT command 182, 183, 534
  $E parameter 460
  Escape sequences 464
  parameters 182
Prompt Message in custom Parameter dialog boxes 91
Properties option 91, 92, 104
Protected mode 228, 262
PRT1 168

Q

QBASIC.EXE
  required for MS-DOS Editor 429
QEMM-386 235, 249
Question mark wild card 46, 272

R

RAM 219, 220, 223, 255, 373, 409
  dynamic 261
  reserved 223
  static 261
RAMDRIVE 409
  compared to SMARTDRIVE 410
RAMDRIVE.SYS 373, 459
  loading into UMB 238
Random Access Memory (see RAM)
RD command (see RMDIR command)
Read-only file attribute 78, 354
  described 79
Read Only Memory (see ROM)
Real mode 228, 262
RECOVER command 340, 379, 393
Redirecting Command Line I/O 166, 169
  using macros 205
Redirection with Doskey macros 205
Refresh option 29, 114
Registers 221, 256
REM command 442, 536
RENAME command 181, 317, 537
  wild cards 317
Rename File dialog box 75, 76
Rename operation 121
  how it works 77
Rename option 44, 52, 75
  multiple selected files 70
Renaming files in Shell 75, 76
Reorder option on Shell File menu 106
Reordering groups in Program List 106
Repaint Screen option 29, 66, 114
REPLACE command 342, 343, 345, 537
Replacing text in Editor 426
Reserve shortcut keys check boxes 101
Reserved memory 222
RESTORE command 322, 539
  ERRORLEVEL values 330
Restore View option 66
RMDIR command 318, 349, 542
ROM 219, 220, 222, 223, 234
  on expansion cards 223
ROM-BIOS 219, 223, 258
Root directory 219-223, 273-276, 281, 358, 368, 371, 389
  cannot be a subdirectory 40
  described 40
  size of 40
Run Dialog Box f85
Run option on Shell File menu 52, 85, 87

S

Savestate (DOSSHELL.INI) 138
Saving Doskey macros 206
Scan codes 13

Screen adjusting in Editor 427
Scroll Bars check box in Editor 427
Scrolling
  pausing 165
  with keyboard 27
  with mouse 27
Search entire disk check box 63
Search File dialog box 63
Search option on Shell File menu 52, 63
Search Results screen f64
Searching in Editor 426
Secondary buffer cache 403
Sectors 346, 363
  sizes 365, 366
  storage of files 367
Segmented addressing 250
Select Across Directories option 56
Select All option 52, 55
Selecting directories in the Shell 42
Selecting files in Shell
  Select All option 55
  with keyboard 54
  with mouse 53
Selection Cursor indicators 26
Semicolon
  command separator in Shell programs 104
SET command 286, 457, 458, 542
SETVER command 543
Shell 21, 23
  Environment (managing) 114
  navigating in 26-31
  overview of 24-25
  startup options 118
SHELL command 458, 459, 544
  different from DOSSHELL 214
Shell programs
  calling batch files 105
  consecutive commands 104
  creating 104
  how they work 104
Shell screen
  redrawing 114
  updating 114
SHIFT command 544
Shift keys 13
Shortcut keys 93, 112
  for Shell programs 92
  icon in this book 4
Show Information dialog box f50
Show Information option 50
Sidekick 227, 240
Single File List view 29
SIZE parameter in DEVICEHIGH command 236, 238
SMARTDRIVE program 407
  compared to RAMDRIVE 410
SMARTDRV.SYS 373, 408, 459
  cannot be loaded into UMB 238, 409
Software defined 5
SORT command 168, 174, 546
  ASCII table as basis 174
Sorting directories with DIR 282
SRAM 261
Static RAM 261
Status Bar in Shell 25
Subdirectories 273, 275, 276, 355
  described 39
  synonymous with "directories" 40
Subdirectory attribute 355
SUBST command 398, 547
  compared to ASSIGN command 400
  incompatibilities 351
  reserving drive letters 484
Super VGA 12, 13
SWITCHES command 548
Switches
  as basic structure of computers 5, 220
  for DOS commands 154
  for DOSSHELL command 118
  stored in DOSSHELL.INI 138
Syntax of DOS Commands 153
SYS command 355, 548
System attribute 332, 355
System files 48, 370, 389
  in File List 48
System file attribute 78, 79

T

Tab Stops setting in Editor 428
Task Swapper 108-114
  effect on exiting Shell 31
  extended memory 100
  limitations 109
  navigating 112
  preventing switching 101
  reserved shortcut keys 101
  shortcut keys 92, 93
  switching on and off 109
  use of memory 99
  using 108
Terminate and Stay Resident programs 215, 227, 235
  copy of DOS environment 243

- DOSKEY 192
- Shell screen 29

Text box in Shell
- adding text 34
- deleting all text 34
- deleting individual characters 34
- using 34

Text files 59
Text mode
- in MS-DOS Shell 115
- differences from graphics mode 116

TIME command 549
Time (including in DOS prompt) 187
Time cycles 478
Time slice 477, 479, 480
Time stamp, changing with COPY 298
Title Bar of Shell 25
Trackball 14
Tracks 363, 367
Tree (see Directory tree) 17
TREE Command 275, 276, 549
- /F switch 276
- MORE filter 277

TSR (see Terminate and Stay Resident programs)
TYPE command 320, 550
Typematic action 13

U

UMB 224, 235
- back-filling 234
- loading device drivers 236
- maximizing 235

Undelete Dialog Box f80
Undelete command 80, 82, 83, 300, 332, 336, 337, 550
- parameters 81

Undelete option 79
Undelete Parameters text box in Shell 81
UNFORMAT command 339, 375, 551
Unformatted files 59
Unix 20
Upper memory 223, 235
Upper memory area 222, 235, 240, 247
Upper Memory Blocks (see UMB)
Upper Memory display of MEM 240

V

VER command 552
VERIFY command 346, 552
Version Number of DOS,
- including in DOS prompt 187

Vertical Bar (|), including in DOS prompt 188
VGA 12, 13, 115, 119
Video displays and cards 10, t12
Video Graphics Array (see VGA)
Video memory 18, 222, 223
Video mode in Shell programs 101
Video mode text box in Shell 101
View Contents option 52
View option on Shell File menu 64
Viewing files in MS-DOS Shell 66
Viewing the Program List 86
Virtual drives 401
VOL command 390, 553
Volume label attribute 355
Volume serial number 303, 389, 390

W

/W switch for EMM386 245
Whole Word check box in Editor 425
Wild cards
- COPY command and 300
- RENAME command and 317
- (see Asterisk and Question mark) 272

Window Title text box in Shell 90
Windows 14, 232, 393
- EMM386 246
- expanded memory 246
- extended memory 250

Words
- 16-bit 5, 220
- 32-bit 5, 220
- defined 5, 220

X

XCOPY command 310, 553
- /A switch 312
- /S switch 313
- ERRORLEVEL values 316
- DISKCOPY compared 301

XGA 12, 115
XMS 231, 246, 257
XMS Memory KB Limit text box 100
XMS Memory KB Required in Shell 100

234

BOOK PROTECTORS & CO.
PROTECTOR HOUSE 76 SOUTH GROVE, LONDON, E17 [illegible]
UK PATENT NOS. [illegible]

Power of... DOS 5.0

This book provides an introduction to the fundamentals of personal computers, and explains the role that DOS 5.0 plays in making the parts of the computer work together.

Power of DOS 5.0 **explains how to customize and control** the DOS 5.0 Shell to provide you with an easy, intuitive access to DOS and the programs on your computer.

Learn how to use DOS 5.0 to help you better manage your computer's memory.

Learn the details of the DOS command line to achieve precise control over the commands you enter.

This informative book explains how to use batch files, which enable you to write your own programs using DOS commands.

TOPICS COVERED INCLUDE:

Introduction to DOS 5.0 • DOS 5.0 Shell • DOS Command Line
Electronic Memory • File Management • Disk Storage
MS-DOS Editor • Batch Files
Configuration Commands • Command Dictionary

Learn to maximize your computing power with DOS 5.0.

ISBN: 1-55828-148-7

9 781558 281486

Tom Goodell is a noted expert in the field of database management systems. He is President and co-founder of Integrated Knowledge Systems, Inc., a Minneapolis-based consulting firm instrumental in the design and development of database and application systems for a long list of clients. His in-depth experience encompasses mainframe, minicomputer, and personal computer environments. Goodell is the author of DOS 4.0: Customizing the Shell (MIS:Press 1989), co-author of R:BASE for DOS (MIS:Press 1988), and the developer of TIMETRACK. In this book, Tom Goodell shows every user how to get the most out of DOS 5.0.

PRICE: $24.95